2015年计算机组成研讨班

多周期CPU形式建模综合方法

多周期数据通路、RTL、时序分析

高小鹏
gxp@buaa.edu.cn

北京航空航天大学计算机学院
2015年7月

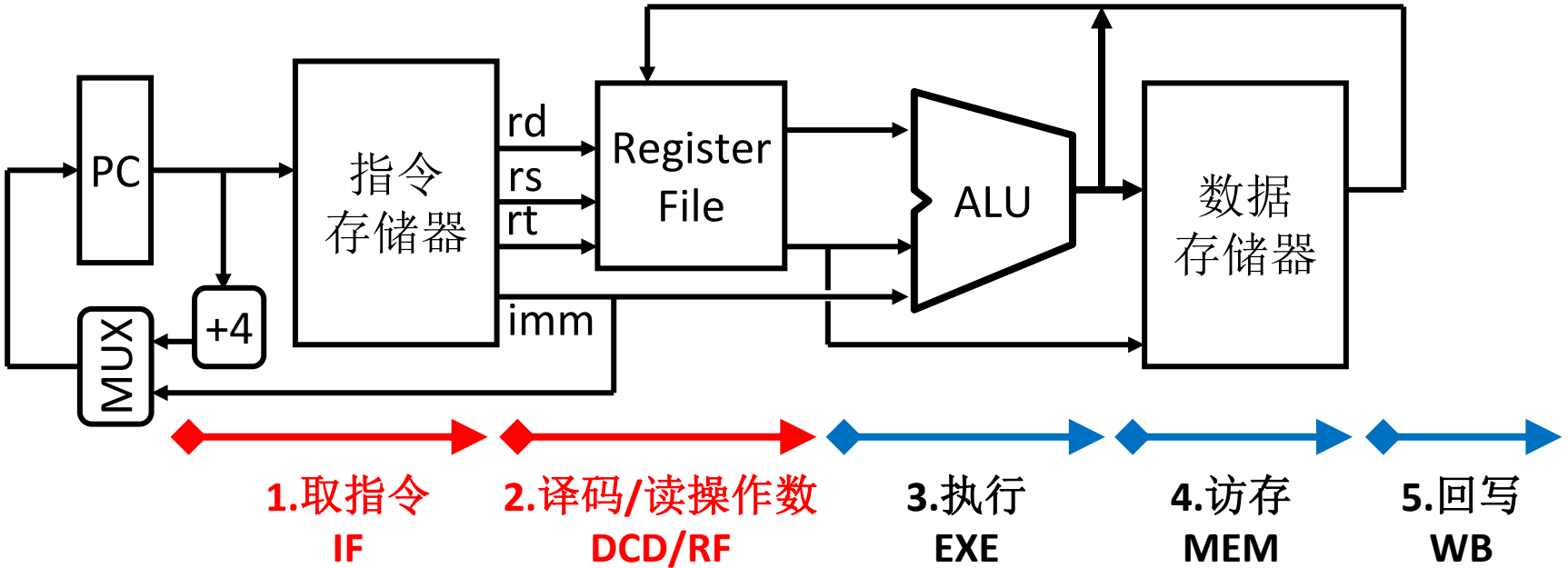
目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

回顾：单周期数据通路

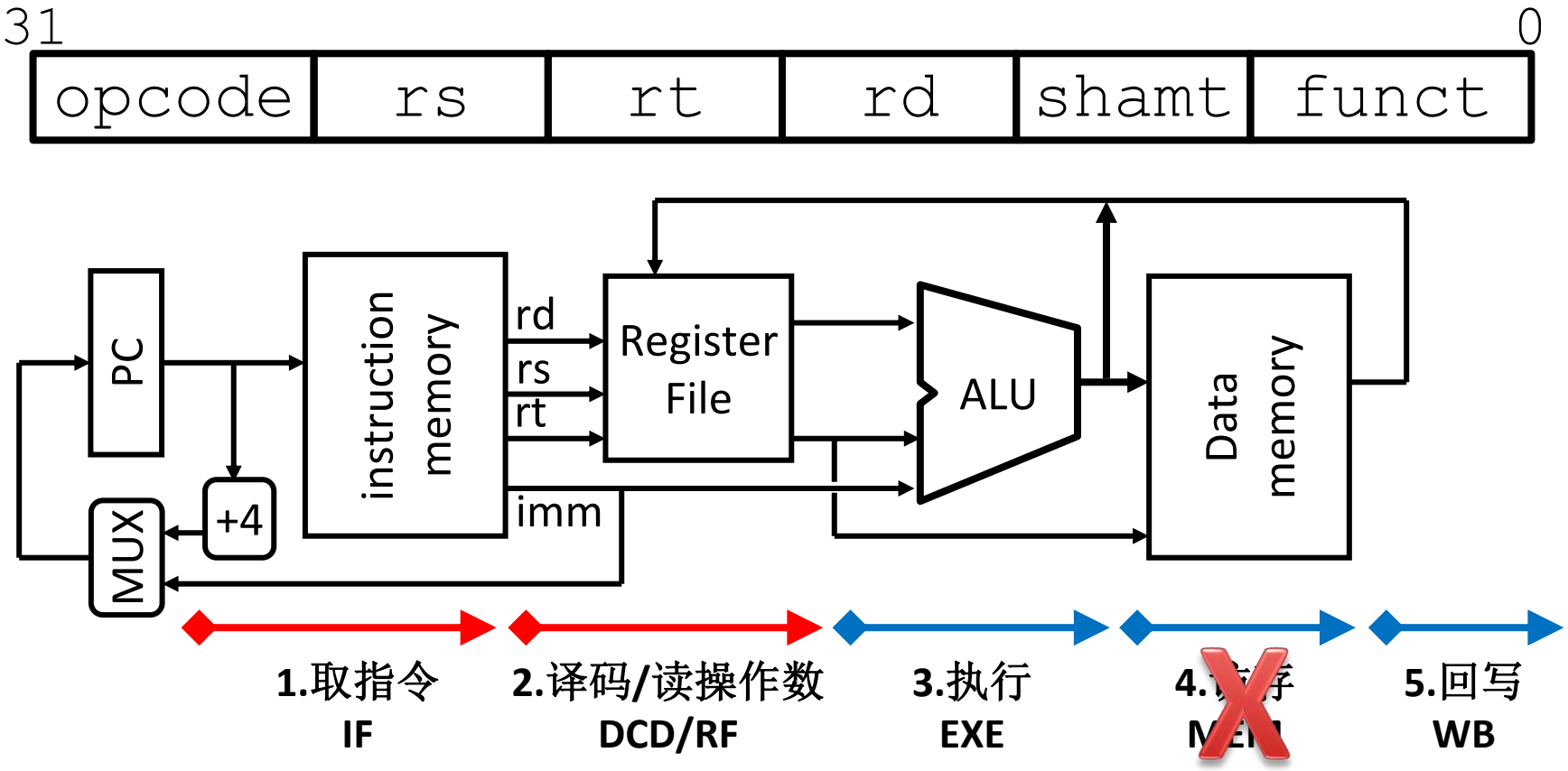
- 从逻辑上：数据通路总共为5个逻辑步骤
- 公共步骤：取指令、译码/读操作数
  - 所有指令均必须经历的2个步骤

取指令	IF	PC驱动IM读取指令
译码/读操作数	DCD/RF	译码属于控制器范畴；可以与读操作数并行
执行	EXE	ALU完成算数/逻辑运算
访存	MEM	读DM或写DM
回写	WB	ALU计算结果或IM读出数据写入寄存器堆



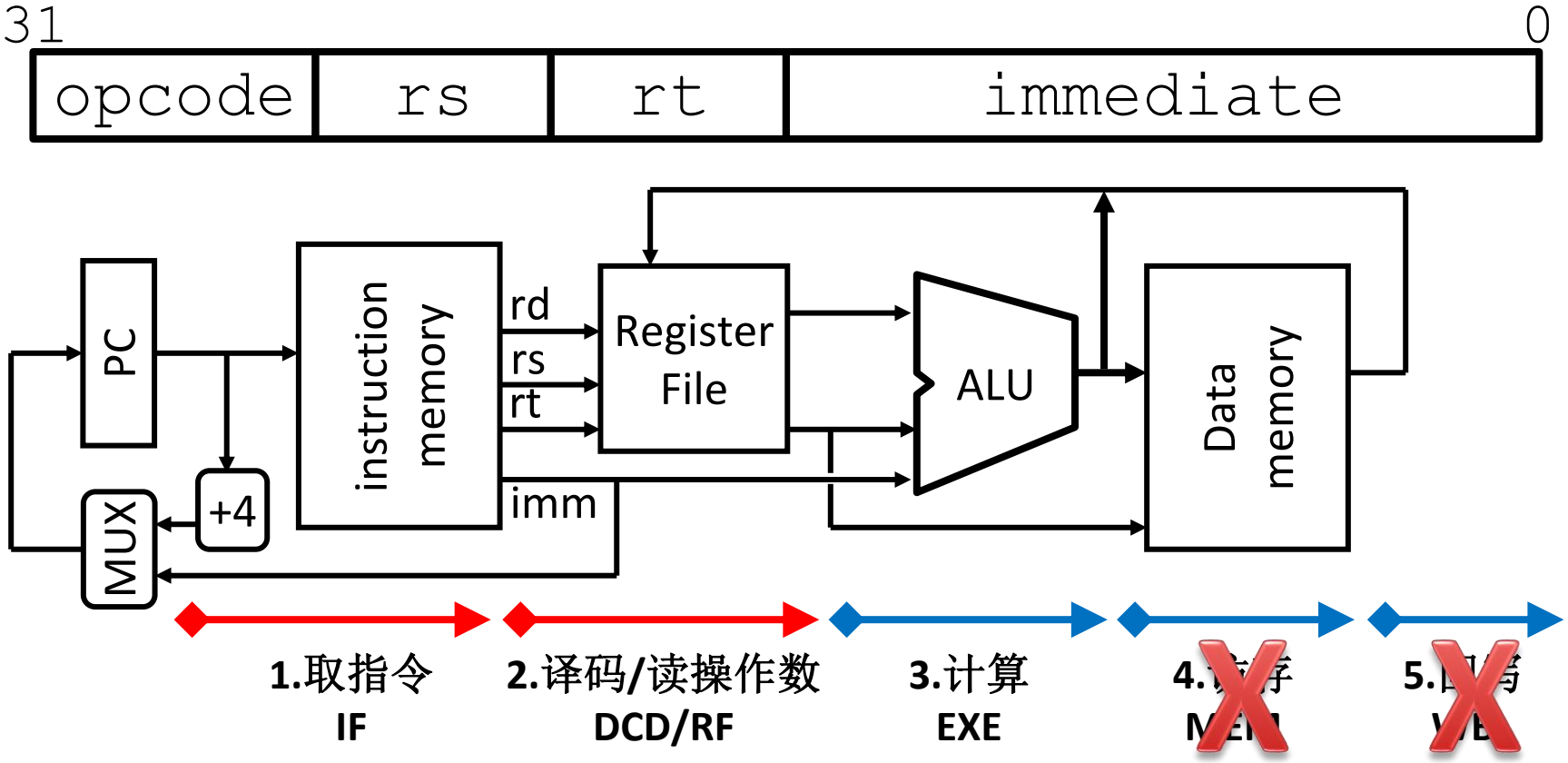
运算类指令：理想执行过程

- 指令：ADD、SUB、OR。。。
- 需求：R[rd] ← R[rs] op R[rt]
- 过程：取指、译码/读寄存器、执行、~~访存~~、回写



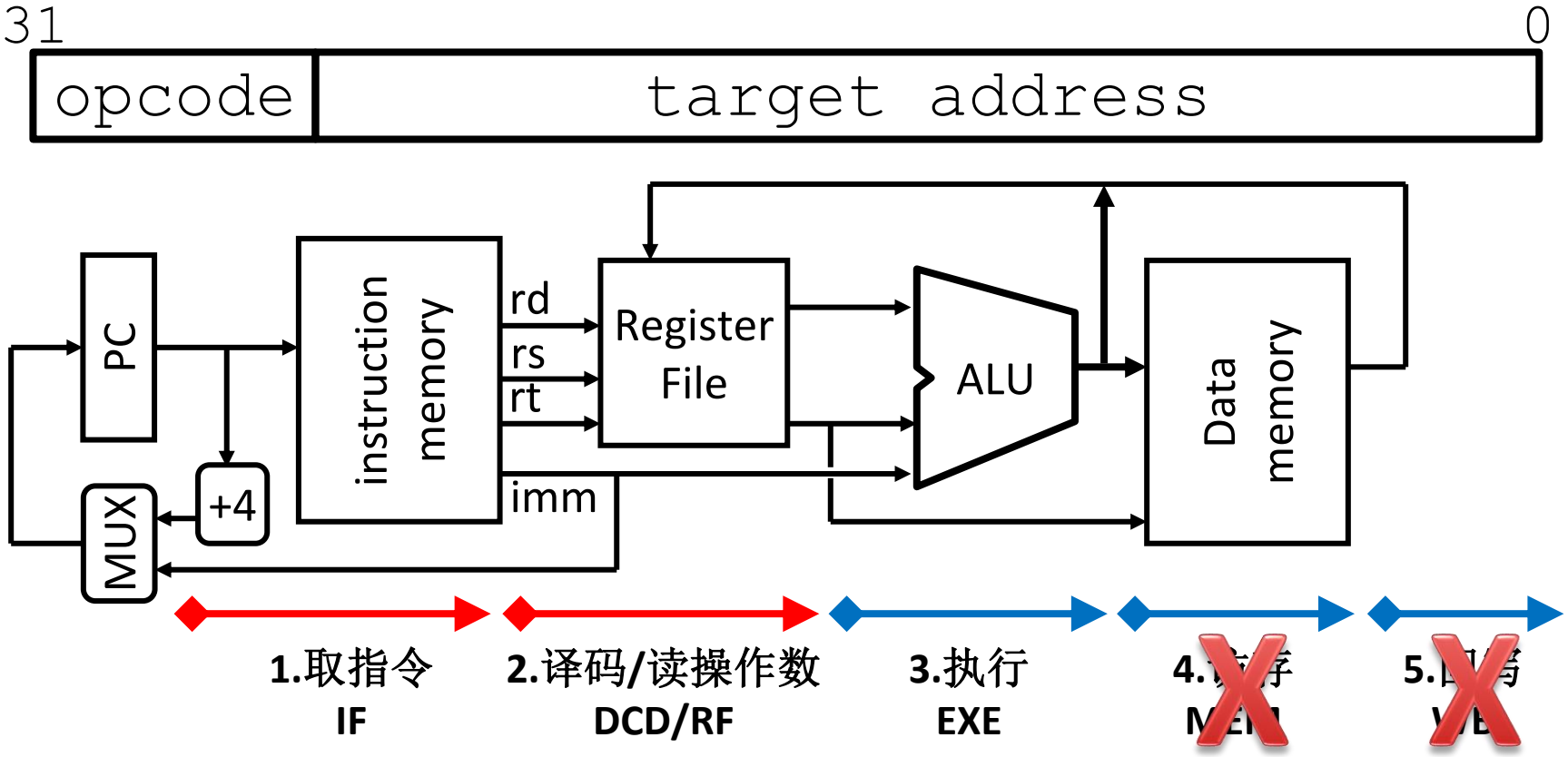
分支类指令：理想执行过程

- 指令：BEQ、。。。
- 需求：PC ← 条件? PC + Ext(Imm) : PC + 4
- 过程：取指、译码/读寄存器、执行、~~访存、回写~~



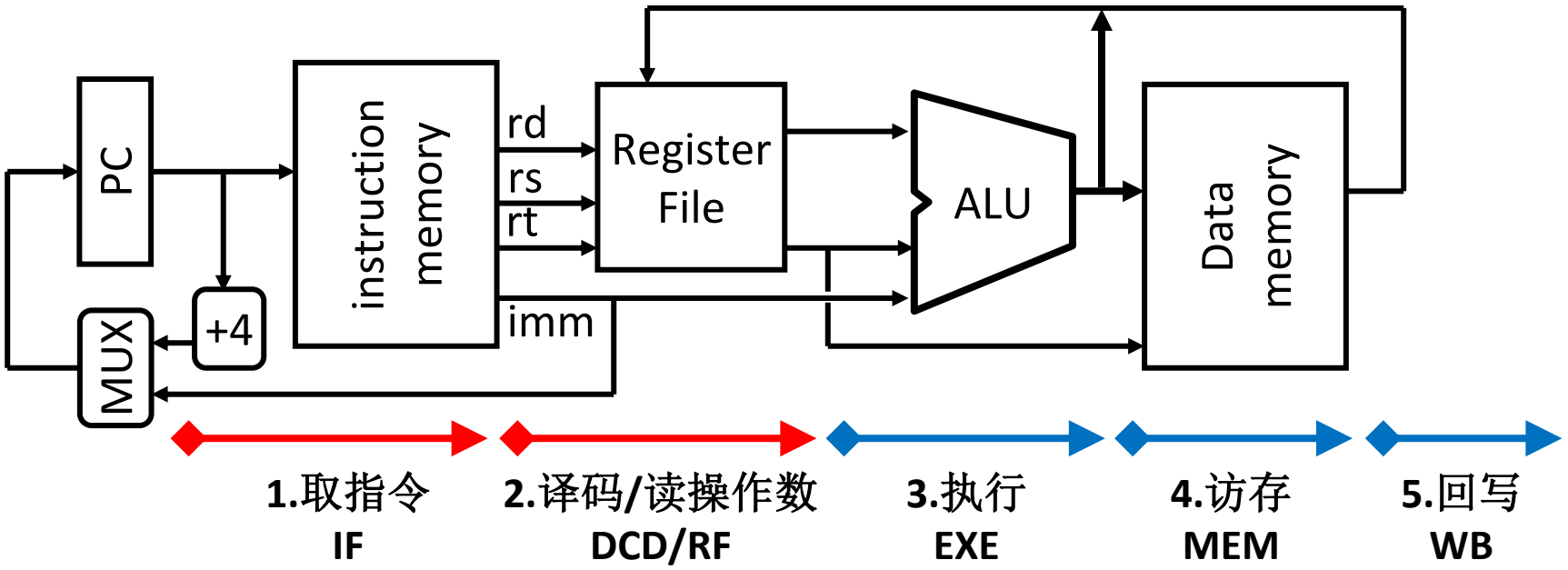
跳转指令：理想执行过程

- 指令：J
- 需求：PC ← PC[31:28] || target_address || 00
- 过程：取指、译码/读寄存器、执行、~~访存~~、~~回写~~



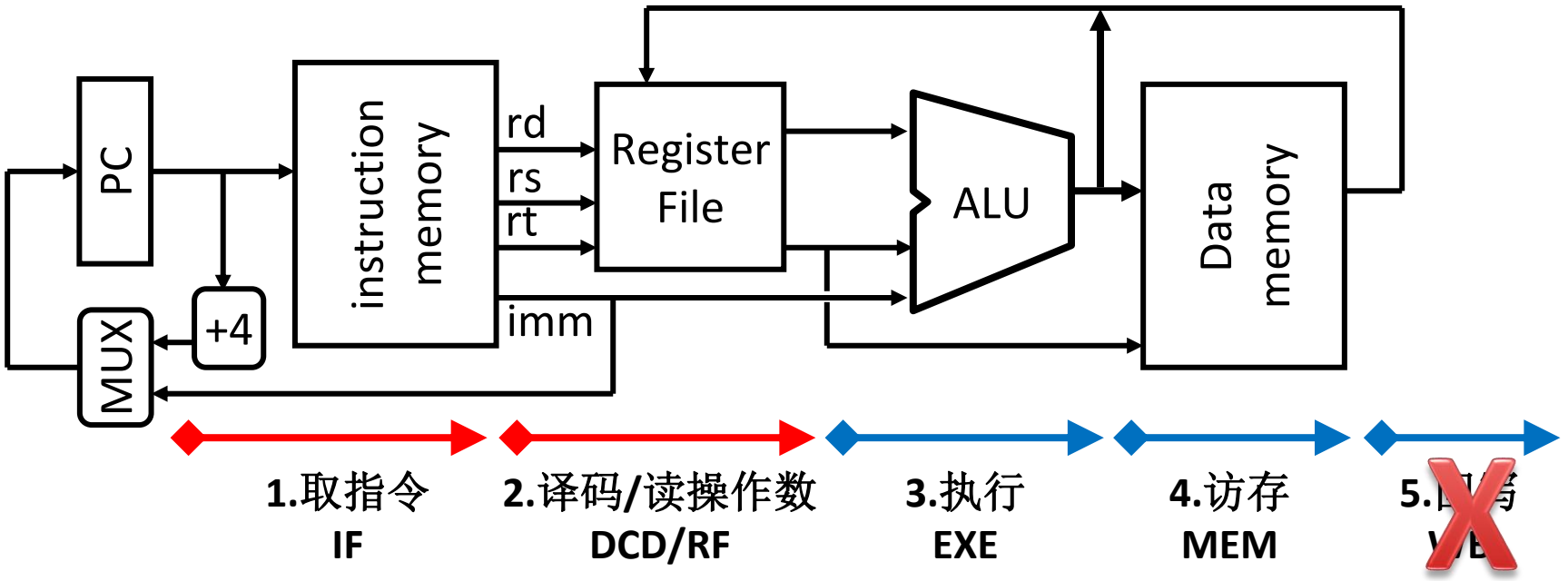
读存储指令：理想执行过程

- 指令：LW
- 需求：RF[rt] ← memory[RF[base] + offset]
- 过程：取指、译码/读寄存器、执行、访存、回写



写存储指令：理想执行过程

- 指令：SW
- 需求：memory[RF[base] + offset] ← RF[rt]
- 过程：取指、译码/读寄存器、执行、访存、~~回写~~



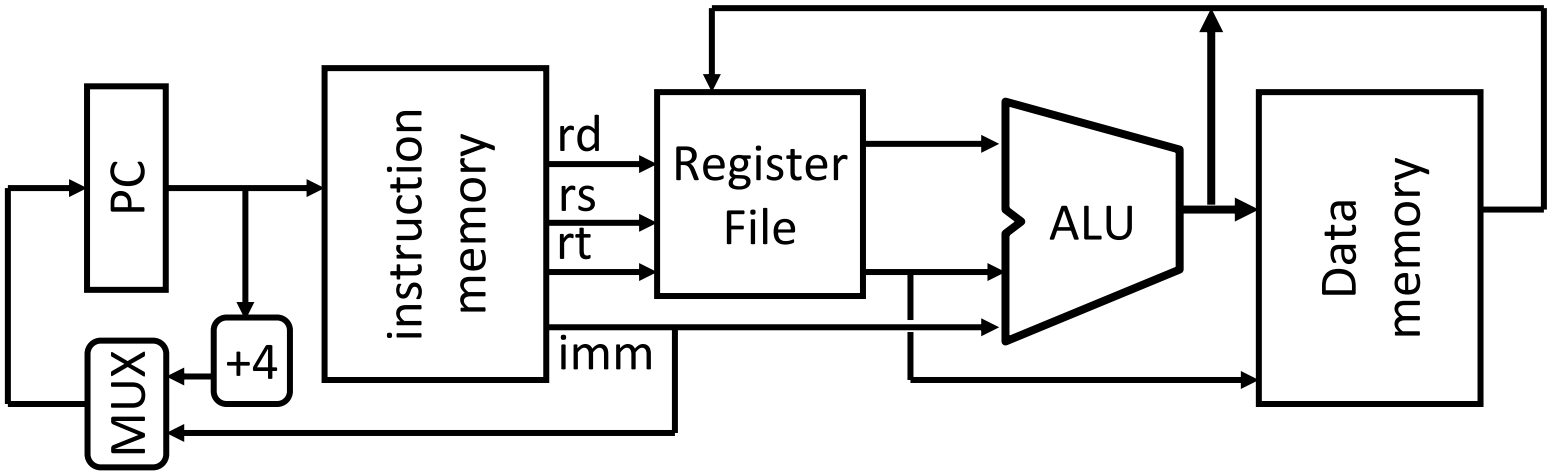
物理执行路径 vs. 理想执行过程

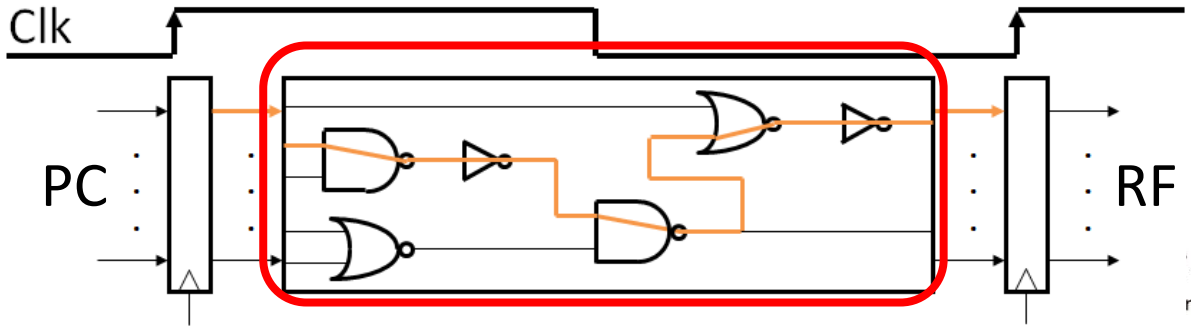
- 2个现象
  - 现象1：不同指令的理想执行过程不同
  - 现象2：所有指令都有前3个阶段
    - 前2阶段完全相同；第3阶段功能有差异
- 启示
  - 不同指令的理想执行过程不同，即理想执行时间不同
  - 不同指令能否具有不同物理执行路径，以对应理想执行过程？

	IF	DCD/RF	EXE	MEM	WB
计算	✓	✓	✓		✓
分支	✓	✓	✓		
跳转	✓	✓	✓		?
读存储	✓	✓	✓	✓	✓
写存储	✓	✓	✓	✓	

单周期数据通路的缺陷

- 模型：PC → 组合逻辑 → 寄存器堆
  - 单一组合逻辑实现了全部5个阶段的逻辑功能
  - 必然存在关键路径，且关键路径导致每条指令延迟均相同
- 结论：无法利用不同指令具有不同执行需求的潜在特性



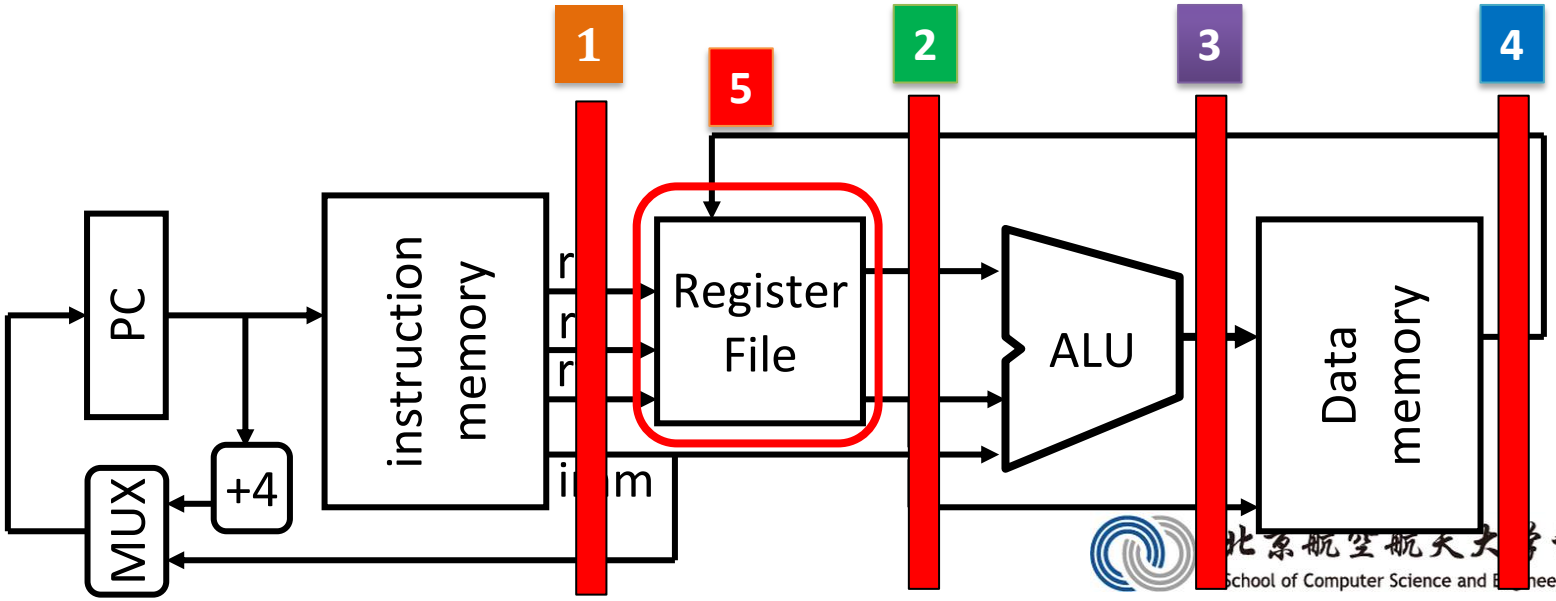


目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

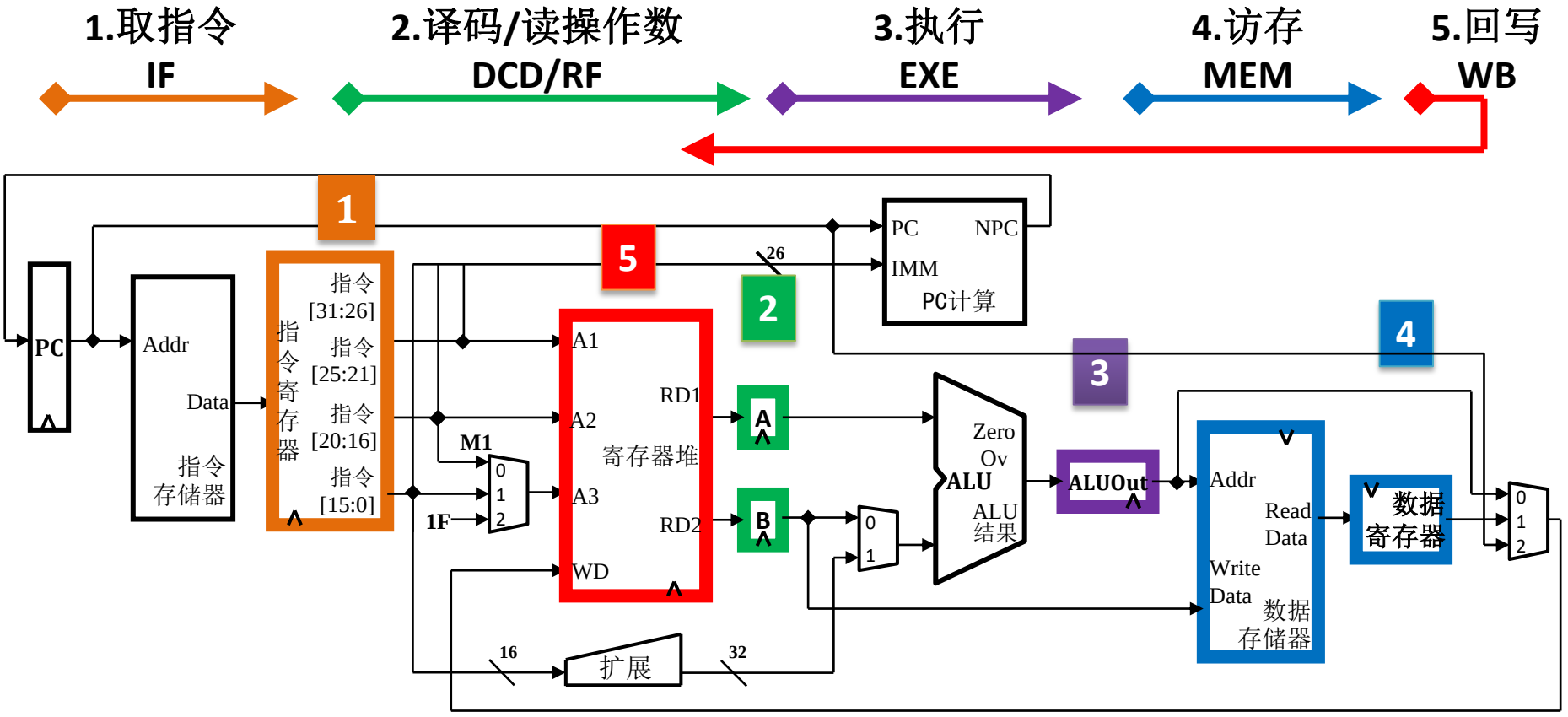
多周期数据通路构思

- 单周期的单一路径被物理切分为多段路径
  - 在数据通路上插入多个寄存器
  - 单一组合逻辑被切分为多段组合逻辑
  - 单一关键路径的大延迟变为多个分段路径的小延迟



基础多周期数据通路

- 支持7条指令
  - ADDU、SUBU、ORI、LW、SW、BEQ、JAL
- PC计算（NPC）
  - 完成与PC相关的一切计算
  - PC+4；PC+imm16；PC+imm26



分段的数据通路

- 5个分段：对应指令的5个可能环节
- 分段结构：读出R-C-写入R(细粒度的R-C-R)
  - 读出R~读出；C~计算；写入R~写入

分段通路	段内 平行功能	读出R	中间逻辑	写入R	依赖
读取指令		PC	IM	IR	
读操作数	读2个寄存器	IR	RF	A/B	读取指令
	读1个寄存器， 立即数扩展	IR	RF → A(/*B*) EXT → 无寄存器？		
执行	R-R运算	A，B	ALU	ALUOut	读操作数
	R-I运算	A，EXT			
访存	读存储	ALUOut	DM	DR	执行
	写存储	ALUOut，B		DM	执行
回写	存储回写	DR		RF	读存储@访存
	计算回写	ALUOut		RF	执行

多周期数据通路特点：功能划分

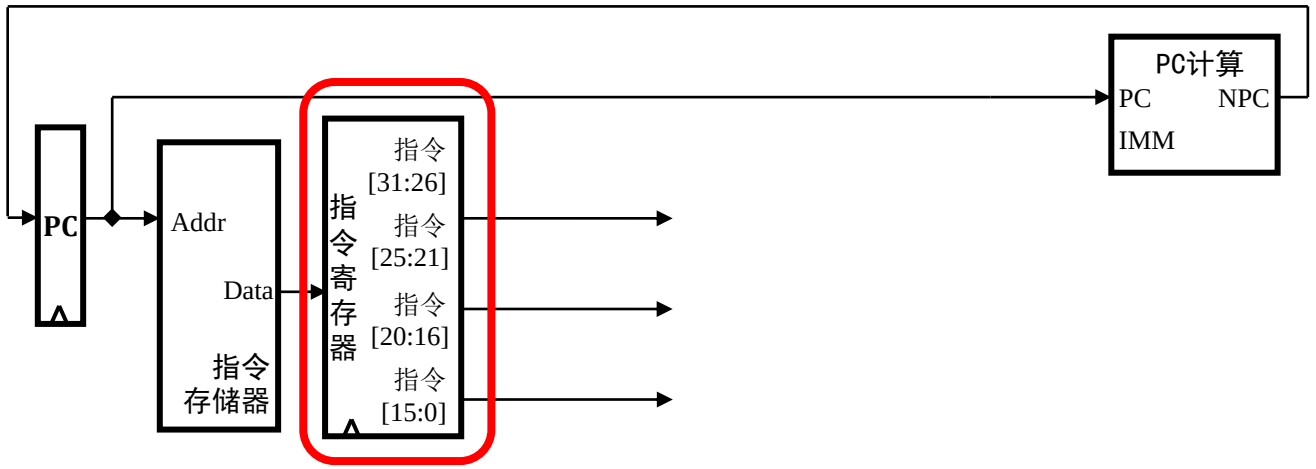
- 把相近的功能部署在同一分段
  - 读操作数：读取RF、立即数扩展
  - 执行：各类算数/逻辑运算
  - 访存：读存储、写存储
  - 回写：ALU计算回写、读存储器回写
- 根据指令需求组合分段，构成相应通路
  - 不同指令执行，占用不同的功能单元
  - 很好的映射不同指令的理想执行过程
  - 共有阶段：阶段1(读取指令)、阶段2(读操作数)

多周期数据通路特点：功能执行

- 段内执行： 每段1个Cycle，数据～处理～保存
  - 数据(读出)：读出R的输出
  - 处理(逻辑)：由本段内的组合逻辑完成，如ALU
  - 保存(写入)：处理的结果保存至写入R
- 段间执行：存在逻辑依赖关系
  - 前段没有执行，后段执行无意义
    - 不是不能执行，而是执行结果无意义
  - 例如：指令不读入IR，读操作数就没有意义！
- 多种周期：分段数量决定cycle数量
  - 与控制器很好的匹配(后面介绍)

ADDU指令的多周期数据通路：IF阶段

ADDU
SUBU
LW
SW
ORI
LUI
BEQ
JAL



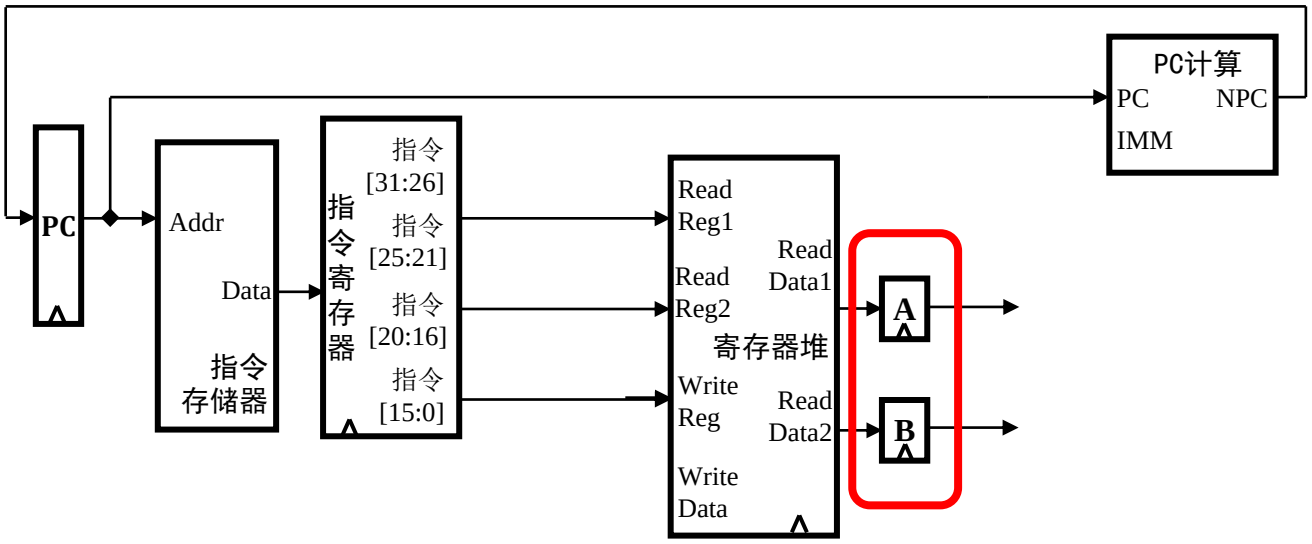
- 加载指令：读取IM，写入IR(指令寄存器)
- 更新PC：PC ← PC + 4

TIP：PC需要写使能

⇨ 与单周期不同了！

ADDU指令的多周期数据通路：DCD/RF阶段

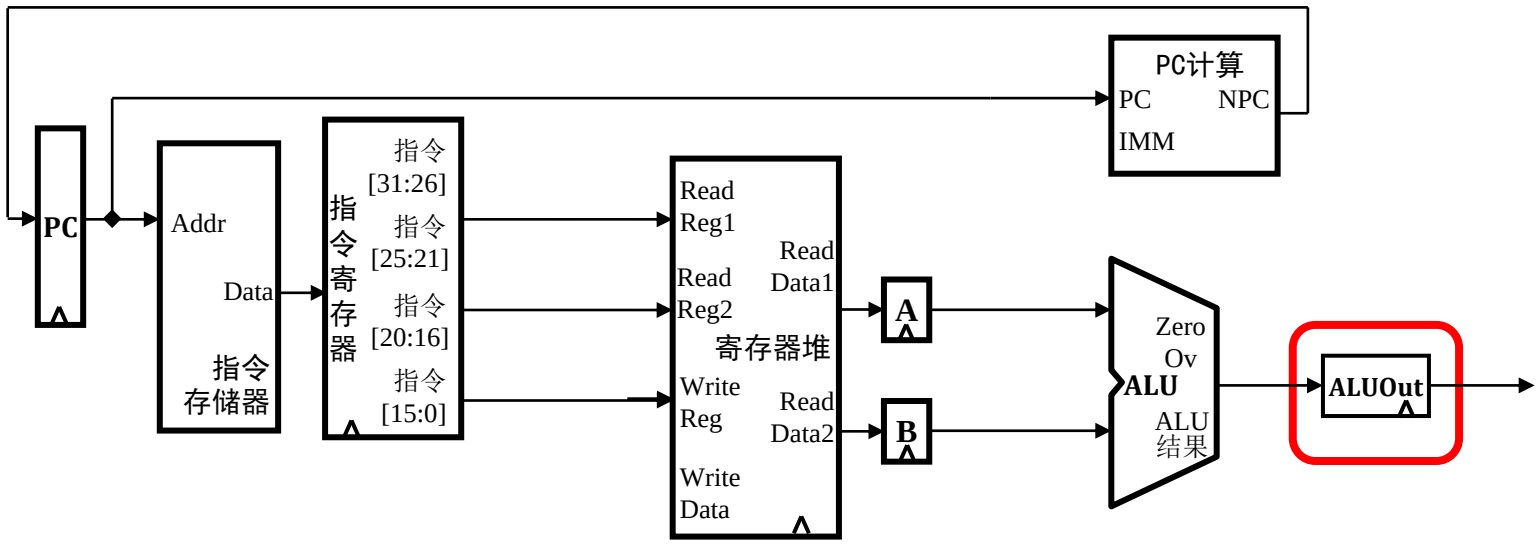
ADDU
SUBU
LW
SW
ORI
LUI
BEQ
JAL



- 读取操作数：RF→A、B
  - A、B：分别存储寄存器文件的2个输出

ADDU指令的多周期数据通路：EXE阶段

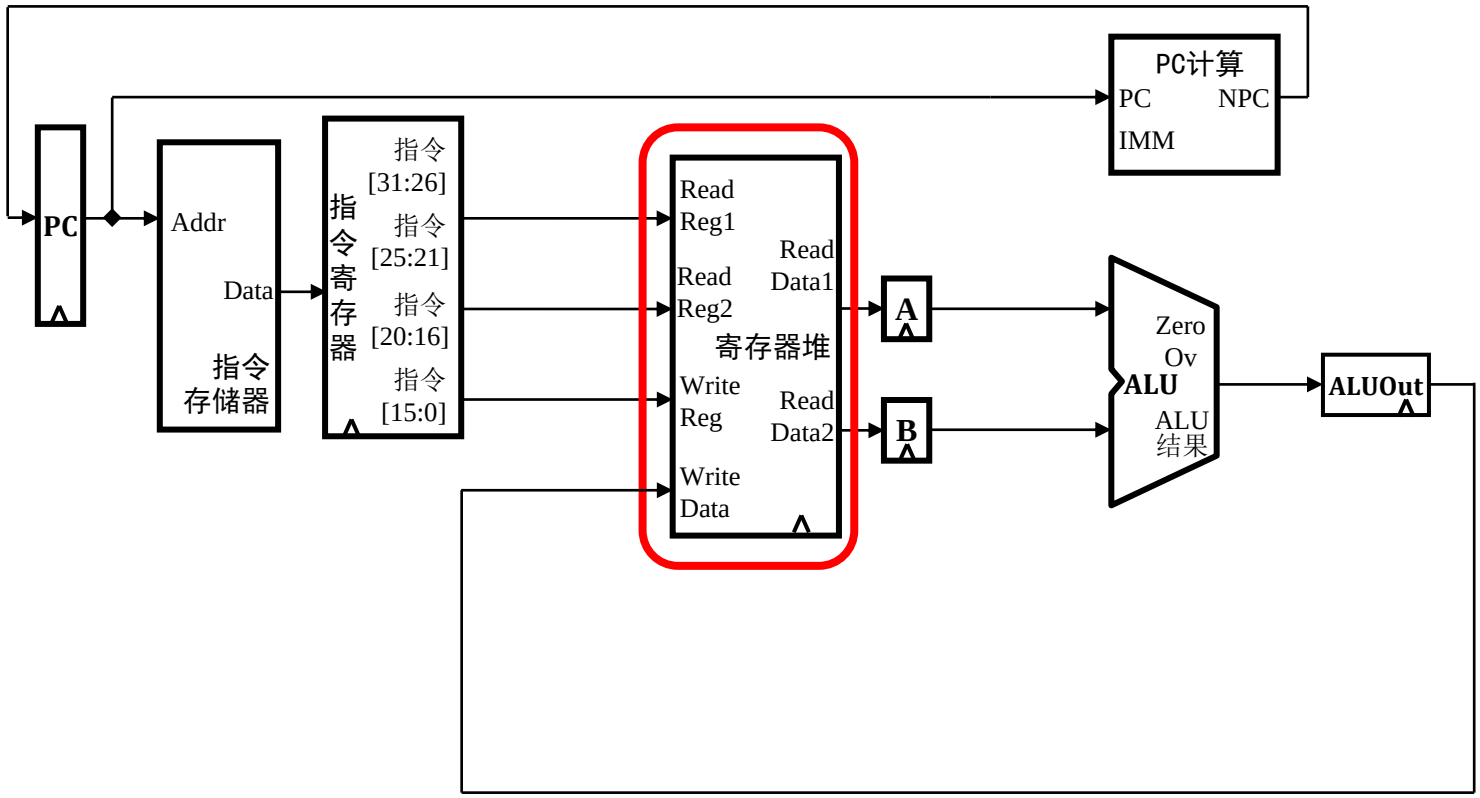
ADDU
SUBU
LW
SW
ORI
LUI
BEQ
JAL



- 计算
  - ALU计算结果存储在ALUOut寄存器

ADDU指令的多周期数据通路：WB阶段

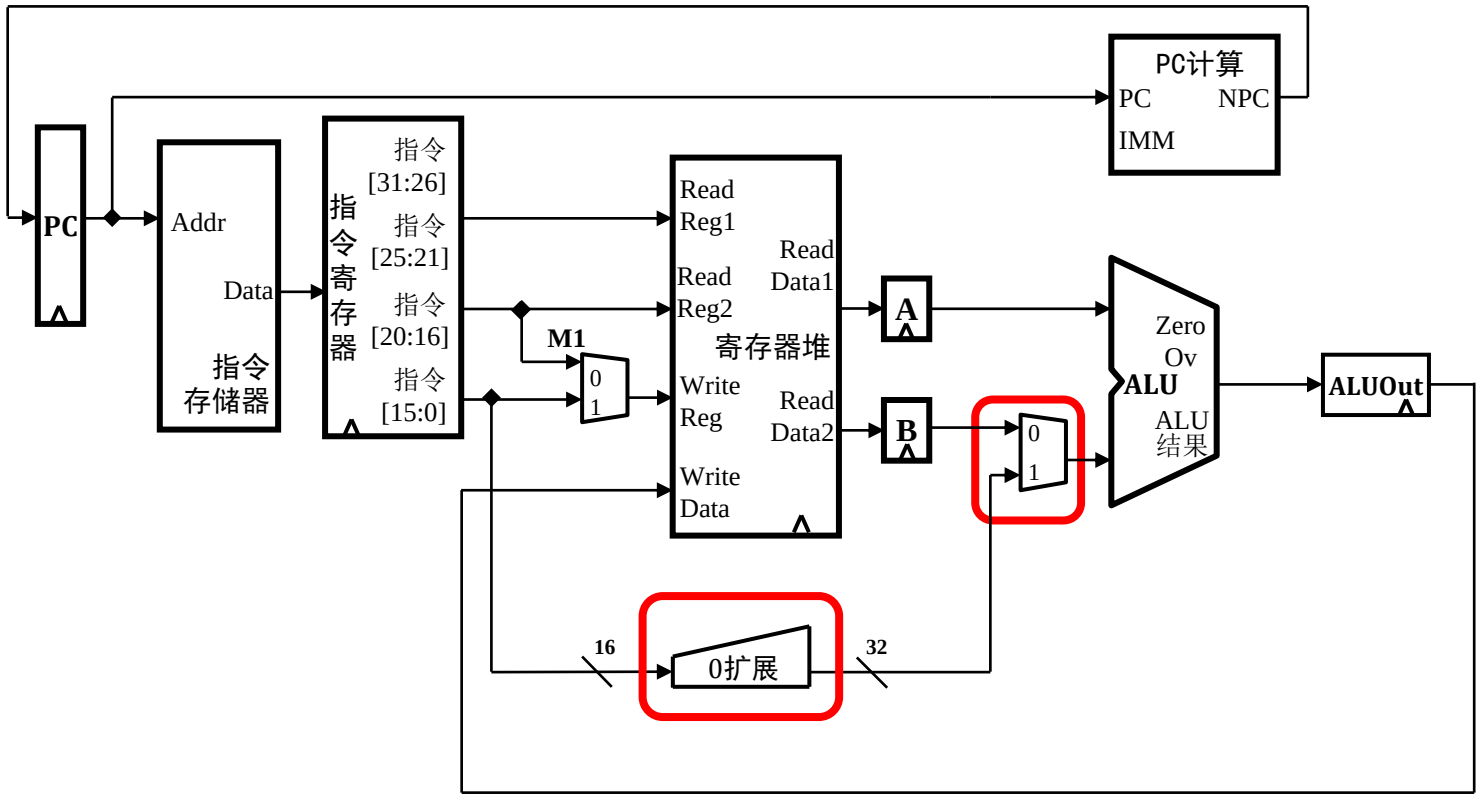
ADDU
SUBU
LW
SW
ORI
LUI
BEQ
JAL



- 回写寄存器
  - ALUOut存储的计算结果写入对应的寄存器

ORI指令的多周期数据通路

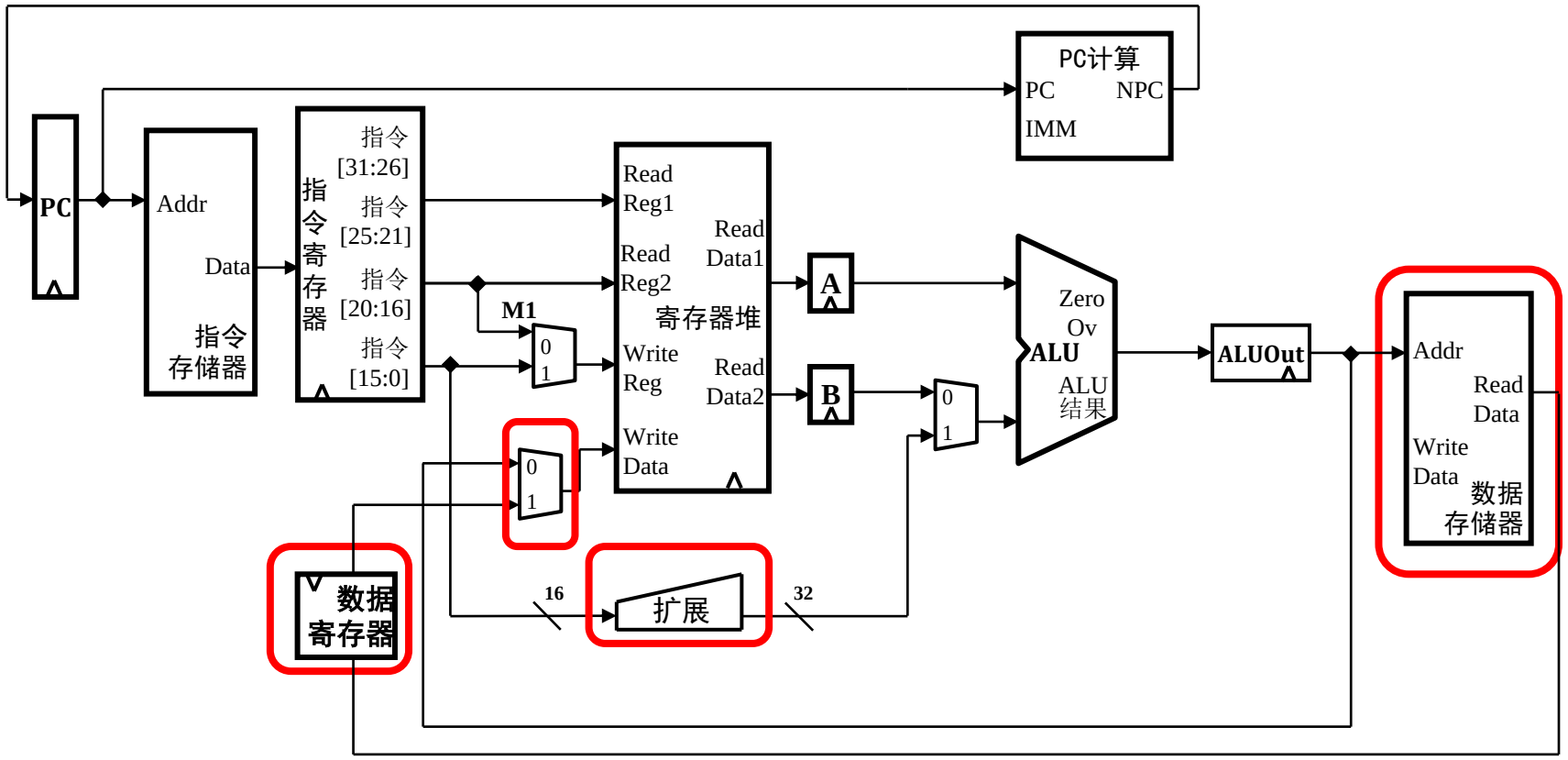
ADDU
SUBU
ORI
LW
SW
BEQ
JAL



- ORI R[rt]←R[rs]|zero_ext(Imm16);
- 增加硬件：零扩展单元、MUX

LW指令的多周期数据通路

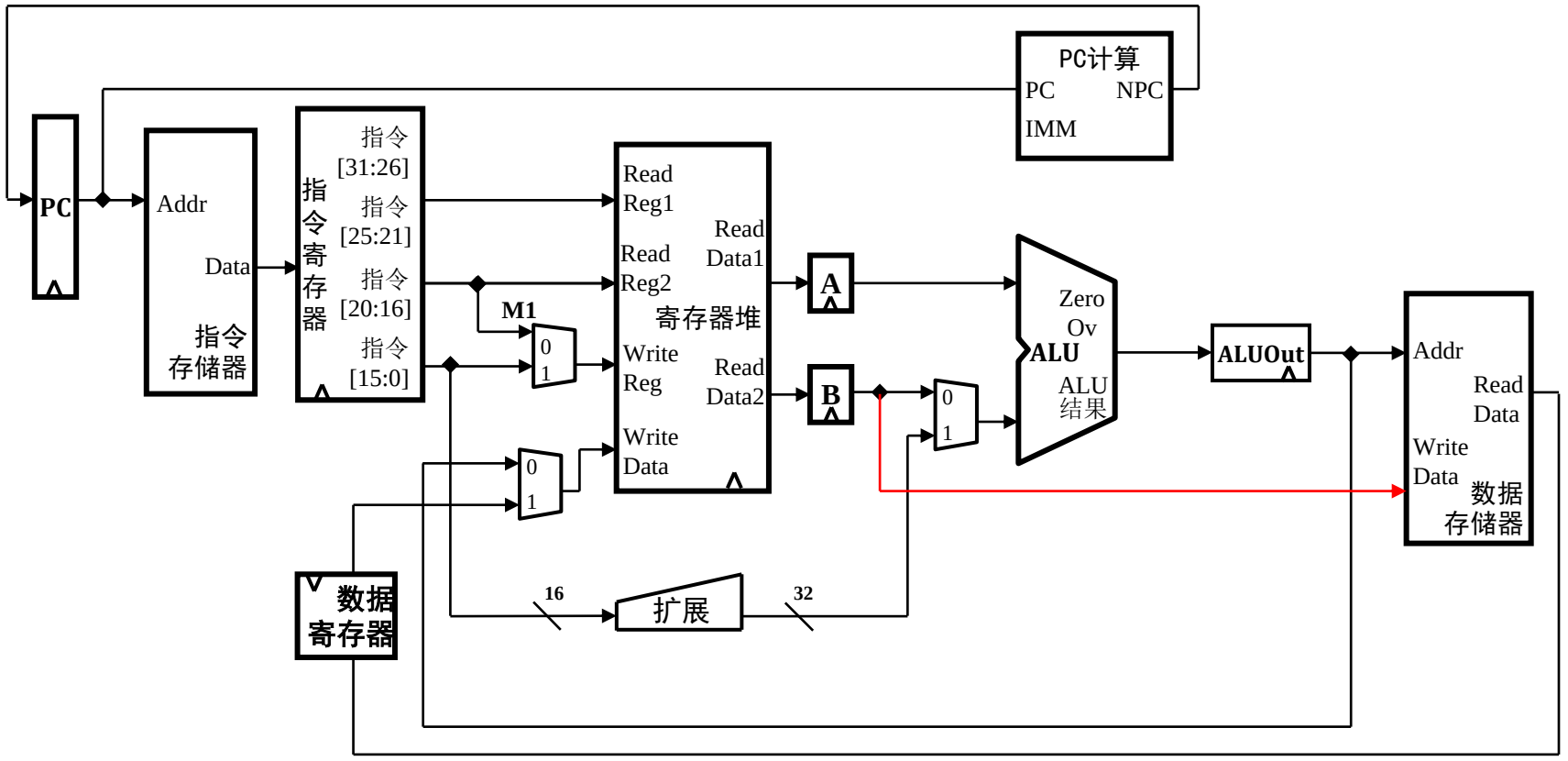
ADDU
SUBU
ORI
LW
SW
BEQ
JAL



- LOAD R[rt]←MEM[R[rs]+sign_ext(Imm16)]
- 增加硬件：数据存储器、数据寄存器、扩展单元、MUX
  - 扩展单元：包括零扩展功能和符合扩展功能

SW指令的多周期数据通路

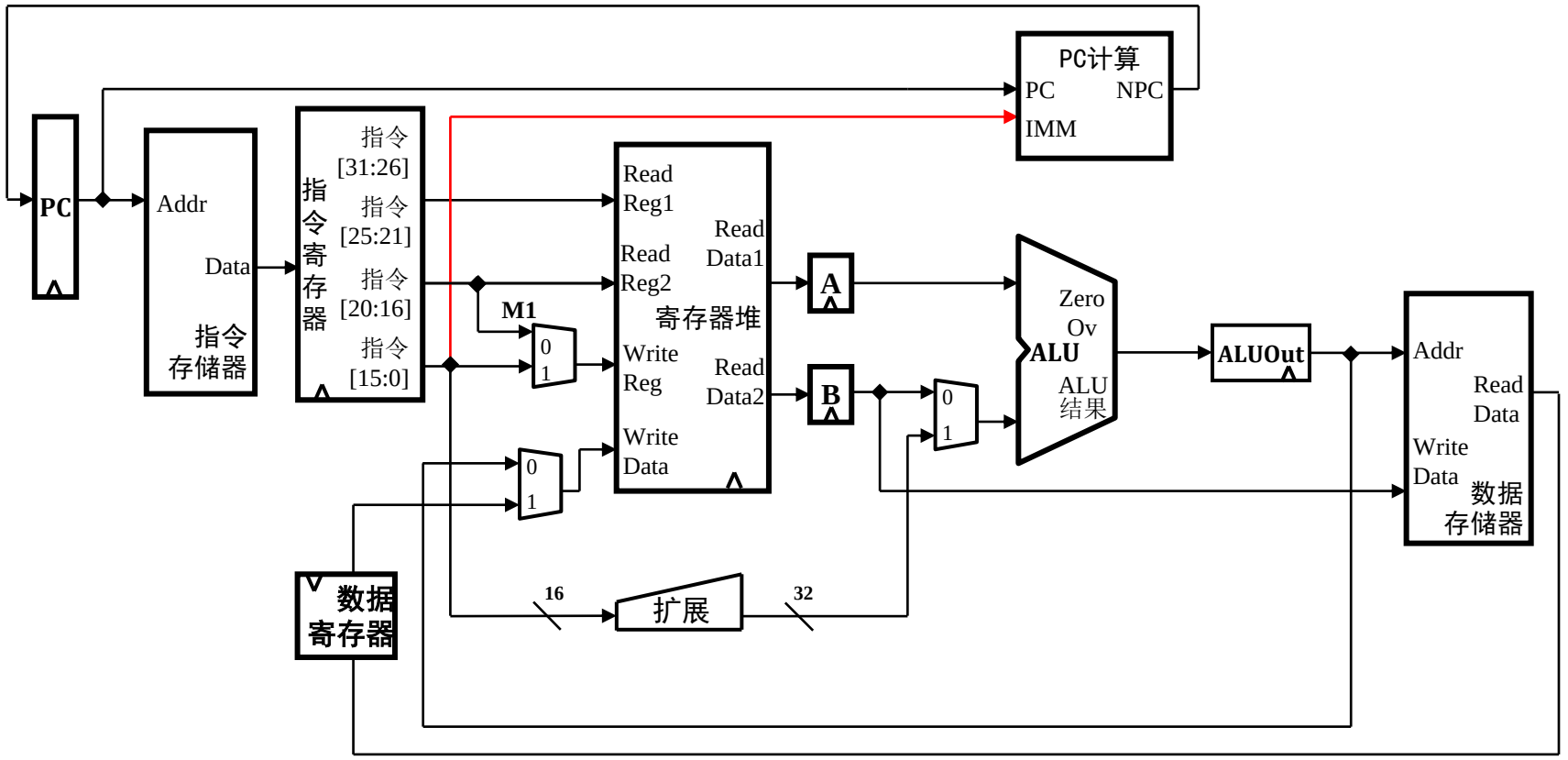
ADDU
SUBU
ORI
LW
SW
BEQ
JAL



- STORE MEM[R[rs]+sign_ext(Imm16)]←R[rt];
- 增加硬件：连接线
  - 从B寄存器至数据存储器的

BEQ指令的多周期数据通路

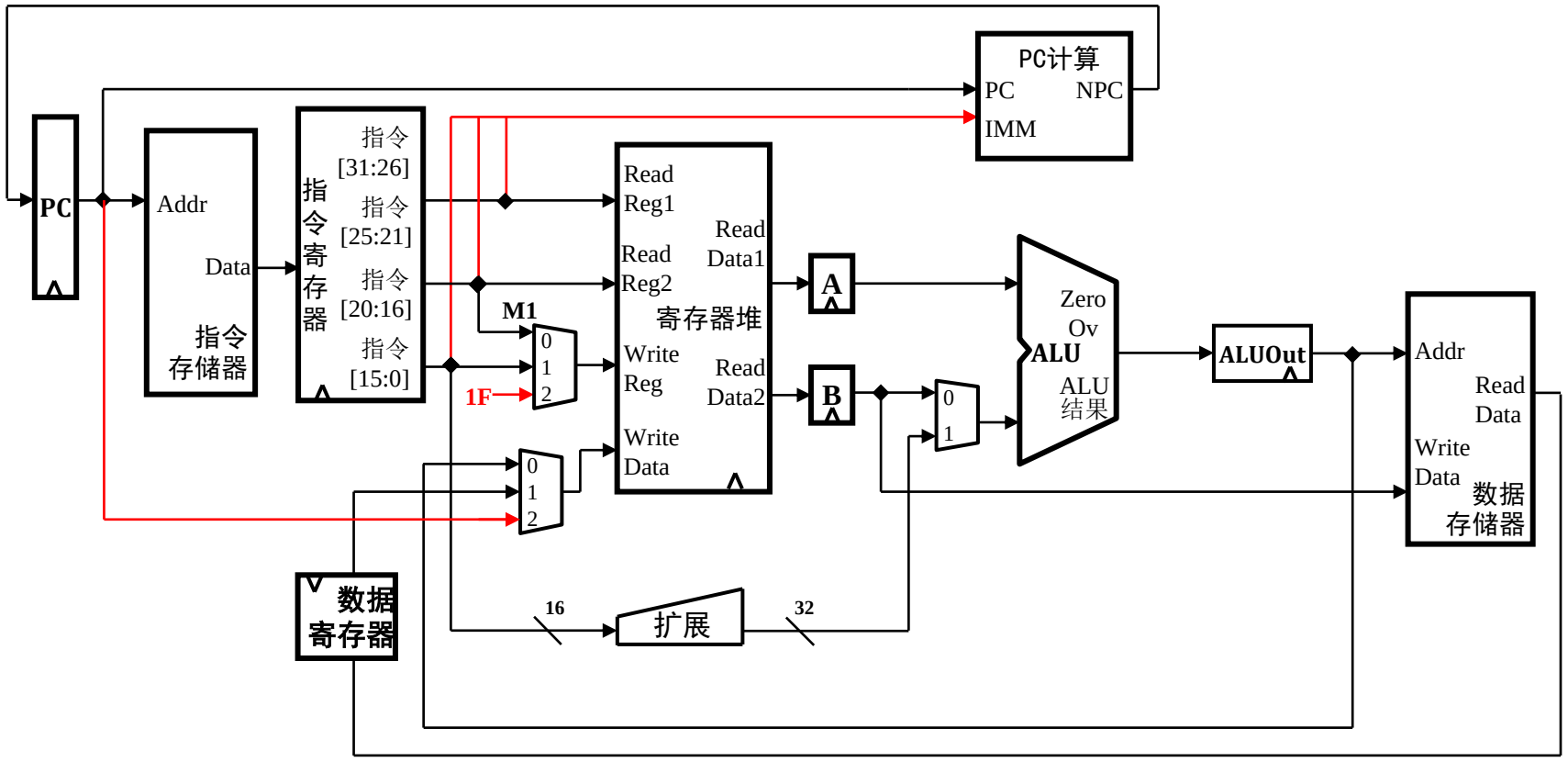
ADDU
SUBU
ORI
LW
SW
BEQ
JAL



- PC ← (R[rs]=R[rt])?PC+sign_ext(imm16):PC+4
- 增加硬件：连接线
  - 从IM[15:0]至PC计算单元

JAL指令的多周期数据通路

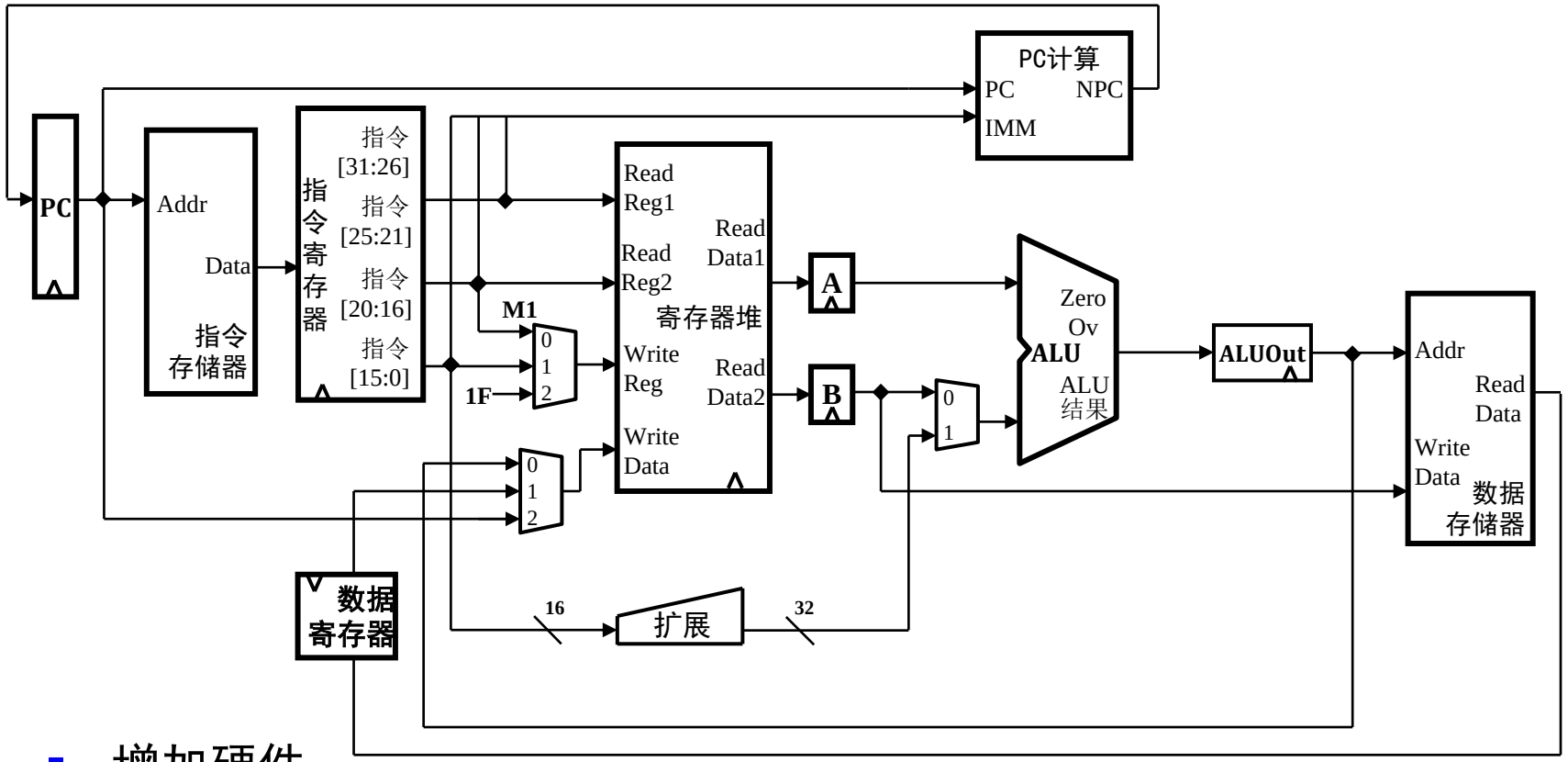
ADDU
SUBU
ORI
LW
SW
BEQ
JAL



- PC ← PC[31:28]||Imm26||00; R[31] ← PC
- 增加硬件：连接线
  - 从IM[15:0]至PC计算单元

多周期数据通路

ADDU
SUBU
ORI
LW
SW
BEQ
JAL



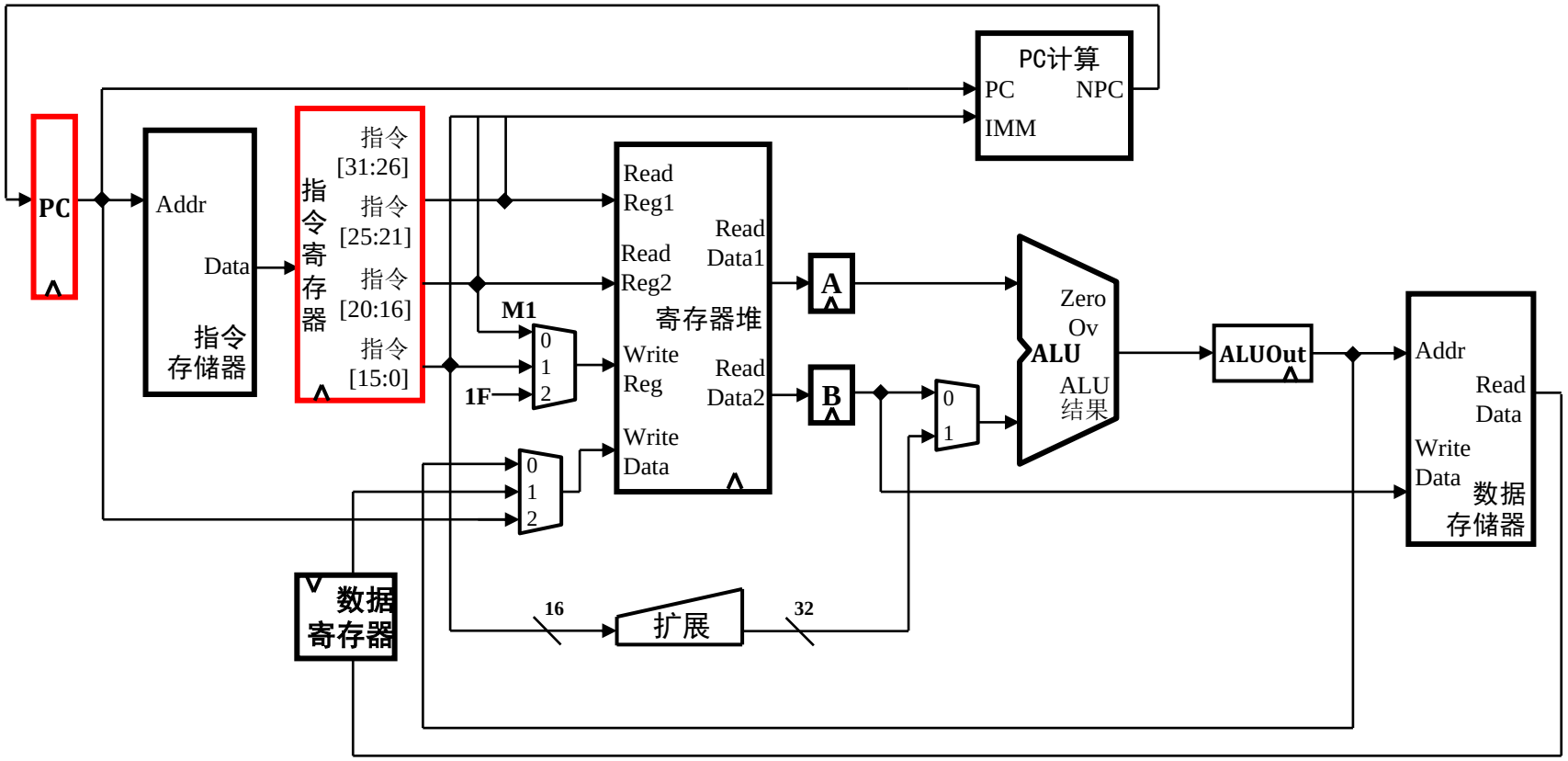
- 增加硬件
  - IR：指令寄存器
  - A/B：操作数寄存器
  - ALUOut：计算结果寄存器
  - DR：数据寄存器

功能部件及其控制信号使用约束

- 注意时钟沿与状态间的关系
  - 寄存器的值：clock正边沿前准备，正边沿写入
- PC、IR、RF、DM：需要写使能
  - PCWr/IRWr/RFWr/DMWr：只能在特定时间有效！其他时间必须无效！
- A/B、ALUOut、DR：不需要写使能
  - 随时可写

分析的要点

- PC不变？
- IR不变？

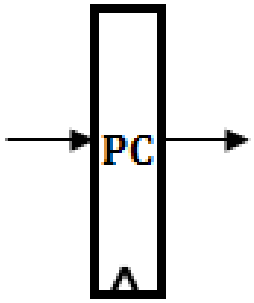


目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

功能部件建模：PC

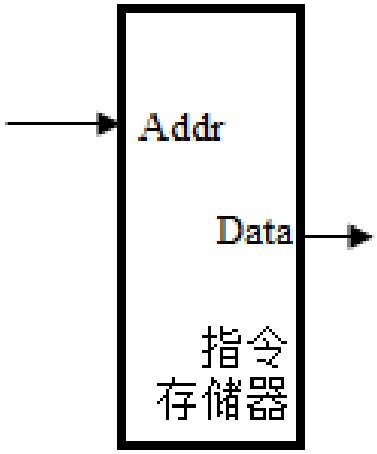
- 程序计数器
- 功能与控制
  - PCWr决定写入



输入	NPC[32:0] PCWr, Clk, RST	
输出	PC[31:2]	
数据结构	addr，30位寄存器	
行为	功能	操作
	计数输出	PC ← addr
	异步复位	if RST then addr ← 32'h40000
	同步加载	Clk上升沿时： if PCWr then addr ← NPC

功能部件建模：IM

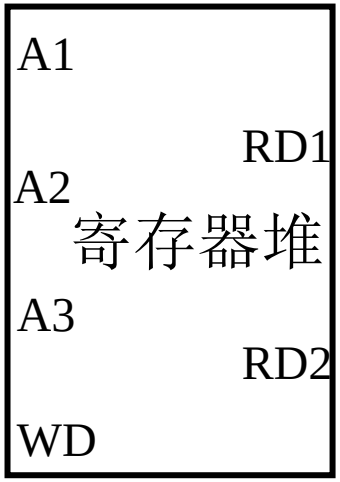
- 数据宽度：32位
- 存储容量：可以任意大
  - 仿真：不超过1K字为宜
- 功能与控制
  - 无需控制，数据经过的固定延迟后输出



输入	AIn[31:0]	
输出	DO[31:0]	
数据结构	Mem为32位×XX字的只读存储器	
行为	功能	操作
	数据输出	DO ← Mem[AIn]

功能部件建模：寄存器堆

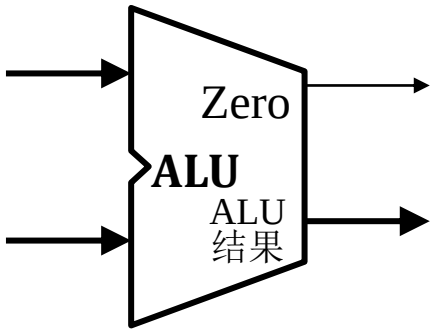
- 功能与控制
  - 读出：不需要控制
  - 写入：RFWr为1



输入	A1[4:0]、A2[4:0]、A3[4:0]、WD[31:0] RFWr、Clk	
输出	RD1[31:0]、RD2[31:0]	
数据结构	RF[0..31]，32个32位寄存器	
行为	功能	操作
	读出寄存器值	RD1←RF[A1]；RD2←RF[A2]
	写入寄存器值	Clk上升沿时 if (RFWr) then RF[A3]←WD

功能部件建模：ALU

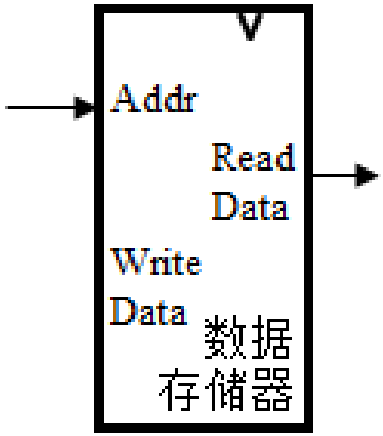
- 功能与控制
  - ALUOp[3:0]决定执行何种计算
- Zero：判断A是否等于B



输入	A[31:0], B[31:0], ALUOp[3:0]		
输出	C[31:0], Zero		
行为	ALUOp	功能	操作
	----	A等于B?	Zero ← (A==B) ? 1 : 0
	0000	加	C ← A + B
	0001	减	C ← A – B
	0010	与	C ← A & B
	0011	或	C ← A | B
	0100	异或	C ← A ^ B
	...	...	...

功能部件建模：DM

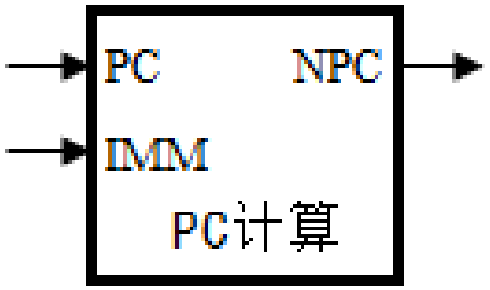
- 数据宽度：32位
- 存储容量：可以任意大
  - 仿真：不超过1K字为宜
- 功能与控制
  - 读出：不需要控制
  - 写入：DMWr为1



输入	AIn[31:0]; WD[31:0] DMWr; Clk	
输出	DO[31:0]	
数据结构	Mem为32位×XX字的存储器	
行为	功能	描述
	读存储	DO ← Mem[AIn]
	写存储	Clk上升沿时 if DMWr then Mem[AIn] ← WD

功能部件建模：NPC

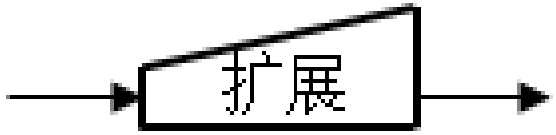
- 计算下条指令的地址
- 功能与控制
  - NPCOp[1:0]决定如何计算PC



输入	PC[31:2], Imm[25:0] NPCOp[1:0]		
输出	NPC[31:2]		
行为	NPCOp	功能	操作
	00	顺序地址	NPC ← PC + 4
	01	计算B指令转移地址	NPC ← PC + sign_ext(imm16) || 00
	10	计算J类指令转移地址	NPC ← PC[31:28] || imm26 || 00
	11	...	...

功能部件建模：EXT

- 立即数扩展
- 功能与控制
  - EXTOp[1:0]决定如何扩展



输入	Imm16[15:0] EXTOp[1:0]		
输出	Imm32[31:0]		
行为	EXTOp	功能	操作
	00	无符号扩展	IMM32 ← {16'b0 || Imm16}
	01	符号扩展	IMM32 ← {{16{Imm16[15]}}, Imm16}
	10	高位扩展	IMM32 ← {Imm16 || 16'b0}
	11		。。。

目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

RTL描述

- RTL： Register Transfer Language
  - 描述CPU在执行指令时其内部的具体步骤(step)
  - 步骤包括操作(operation)、寄存器间的通信以及步骤见的时序关系
- 步骤：在1个cycle内完成的操作集合
- 描述方法：
  - 串行描述：step1、step2、step3是顺序执行，每个step执行1个cycle
  - 并行描述：op2、op3是同时执行，并在同一个cycle完成

```
step1: op1
step2: op2; op3
step3: op4
```

建立指令的多周期RTL描述的基本方法

- 每条指令映射到3～5个cycle
- 重点刻画部件间的连接关系及部件应执行的功能
- 有依赖关系的operation必须部署在不同的cycle
  - 这是由于分段数据通路的依赖性决定的
- 可以提早执行的operation就“尽早执行”
  - 源于门电路的并行性特点；即使后面用不上也无所谓
- 公共cycle：任何指令都必须包括
  - Cycle1：读取指令，以指令写入IR为结束
  - Cycle2：准备操作数，以操作数写入A/B为结束(也包括立即数扩展环节)
    - 操作数：包括从RF中读取寄存器、立即数扩展

ADDU：RTL建模分析

R[rd]←R[rs]+R[rt]

- Cycle1：Fetch(读取指令)
  - Op1：IM中读出的数据(即指令)写入IR
  - Op2：计算PC=PC+4(PC指向下条指令)
    - Op2部署在同一个cycle的理由：硬件设计基本原则之一是“尽早执行”
    - 即便后续操作需要再次改变PC也无所谓
  - 注意1：由于寄存器时序特点，因此Op2不影响Op1
  - 注意2：所有指令都必须包括该step

ADDU：RTL建模分析

R[rd]←R[rs]+R[rt]

- Cycle2：DCD/RF(读取操作数)
  - Op1和Op2分别把RS和RT写入A、B
  - 所有指令都必须包括该step
    - 该周期隐含包括另一重要步骤：指令译码(控制器内完成)
- Cycle3：Exe(执行)
  - Op：ALU完成计算并写入ALUOut
- Cycle4：ALUWB(结果回写)
  - Op：ALUOut写入RF的RD寄存器

ADDU：RTL描述表

R[rd]←R[rs]+R[rt]

- 4个cycle
  - →IR→A/B→ALUOut→RF
- RTL描述表
  - 功能部件：在该cycle需要控制的功能部件
  - 控制信号：功能部件的控制信号在该cycle的取值

周期序号	周期名称	语义	RTL	功能部件	控制信号
Cycle1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr←1; NPCOp←+4; PCWr←1
Cycle2	DCD/RF 读操作数	2个操作数存入A/B	A←RF[rs]; B←RF[rt]		
Cycle3	Exe 执行	执行加法，结果存入ALUOut	ALUOut←ALU(A,B)	ALU	ALUOp←ADD
Cycle4	AluWB 回写	计算结果回写至rd寄存器	RF[rd]←ALUOut	RF	RFWr←1

LW：RTL描述表

- 5个周期

R[rt]←MEM[R[rs]+sign_ext(imm16)]

  - →IR→A/Ext→ALUOut→DR→RF

周期	步骤	语义	RTL	需控制的功能部件	功能部件控制信号
Cycle1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr←1; NPCOp←+4; PCWr←1
Cycle2	DCD/RF 读操作数	基地址存入A； 偏移符号扩展	A←RF[rs] ***EXT(Imm16)***	EXT	EXTOp←SE
Cycle3	MA 计算地址	执行加法，结果存入ALUOut	ALUOut←ALU(A,EXT)	ALU	EXTOp←SE ALUOp←ADD
Cycle4	MR 读存储器	读取DM，数据存储DR	DR←DM[ALUOut]		
Cycle5	MemWB 回写	DR写入rt寄存器	RF[rt]←DR	RF	RFWr←1

SW：RTL描述表

- 4个周期

MEM[R[rs]+sign_ext(imm16)] ← R[rt]

  - →IR→A/Ext→ALUOut→DM

周期	步骤	语义	RTL	需控制的功能部件	控制信号
Cycle1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr←1 NPCOp←+4 PCWr←1
Cycle2	DCD/RF 读操作数	基地址存入A； 偏移符号扩展	A←RF[rs] EXT(Imm16)	EXT	EXTOp←SE
Cycle3	MA 计算地址	执行加法，结果存入ALUOut	ALUOut←ALU(A,EXT)	ALU	EXTOp←SE ALUOp←ADD
Cycle4	MW 写存储器	rt寄存器写入DM	DM[ALUOut]←RF[rt]	DM	DMWr←1

ORI：RTL描述表

R[rt]←R[rs]+zero_ext(imm16)

- 4个cycle
  - →IR→A/EXT→ALUOut→RF

周期序号	周期名称	语义	RTL	需控制的功能部件	功能部件控制信号
Cycle1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr←1 NPCOp←+4 PCWr←1
Cycle2	DCD/RF 读操作数	操作数存入A 无符号扩展	A←RF[rs]; EXT(Imm16)	EXT	EXTOp←UE
Cycle3	Exe 执行	执行加法，结果存入ALUOut	ALUOut←ALU(A,EXT)	ALU	EXTOp←UE ALUOp←OR
Cycle4	AluWB 回写	计算结果回写至rt寄存器	RF[rt]←ALUOut	RF	RFWr←1

BEQ：RTL描述表

```
PC ← (R[rs]==R[rt]) ?
     PC+4+(sign_ext(imm16)||00) :
     PC+4
```

- 3个cycle
  - →IR→A/EXT→PC
- 所有PC计算都在NPC中完成
  - PC+4：cycle1
  - PC+偏移：cycle3

周期序号	周期名称	语义	RTL	需控制的功能部件	功能部件控制信号
Cycle1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr←1 NPCOp←+4 PCWr←1
Cycle2	DCD/RF 读操作数	RS操作数存入A; 32位无符号扩展	A←RF[rs]; B←RF[rt]		
Cycle3	Br 执行	执行减法，判断Zero	ALUOut←ALU(A,B) PC←NPC(PC,imm16,Zero)	ALU NPC PC	ALUOp←SUB NPC←BNPC PCWr←Zero

JAL：RTL描述表

PC ← PC[31:28] || imm26 || 00
R[31] ← PC

- 3个cycle
  - →IR→空→PC
- 空：为了不使得cycle2的语义过于复杂
- 优化设计：2个cycle！

周期序号	周期名称	语义	RTL	功能部件	功能部件控制信号
Cycle1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr←1 NPCOp←+4 PCWr←1
Cycle2	DCD/RF 读操作数				
Cycle3	JMP 执行	计算并保存转移PC; 保存PC	RF[31]←PC PC←NPC(PC,imm26)	RF NPC PC	RFWr←1 NPCOp←JNPC PCWr←1

LUI：RTL描述表

$R[rt] \leftarrow imm16 \,||\, 0^{16}$

- 4个cycle
  - →IR→A/EXT→ALUOut→RF
- 借用R型计算
  - 注意$0的用途：可以体会MIPS指令格式定义的巧妙

Q：可否减少cycle？

周期序号	周期名称	语义	RTL	需控制的功能部件	功能部件控制信号
Cycle1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr←1 NPCOp←+4 PCWr←1
Cycle2	DCD/RF 读操作数	$0存入A 无符号扩展	A←RF[0] EXT(Imm16)	EXT	EXTOp←HE
Cycle3	Exe 执行	执行加法，结果存入ALUOut	ALUOut←ALU(A,EXT)	ALU	EXTOp←HE ALUOp←OR
Cycle4	AluWB 回写	计算结果回写至rt寄存器	RF[rt]←ALUOut	RF	RFWr←1

目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

时序分析的目的

- 1、时序是理解难点之一
  - 理解不正确，设计/调试就易于出错
  - 虽然很简单(时钟沿之前建立，时钟沿后存储)，但往往理解不正确
- 2、有利于后续状态机的设计与理解
  - 更好的理解寄存器的数据准备与数据写入的关系
- 3、多周期时序复杂度适中，适于讲解时序
  - 变化：PC、IR、A/B、DR、RF、DM会发生变化
  - 稳定：在1条指令内只变化1次(PC除外)
- 4、进一步训练学生学习形式建模的方法，提高抽象思维能力

时序分析要点：RTL制导

- RTL制导分析需要哪些必要环节
  - 并非所有环节都需要(多周期的基本特点)
  - 只需关注前序寄存器与后继寄存器间关系
    - 前序值在时钟沿到来后写入寄存器
- ☞注意时钟周期的概念
  - 2个时钟边沿之间的时间

LW的RTL

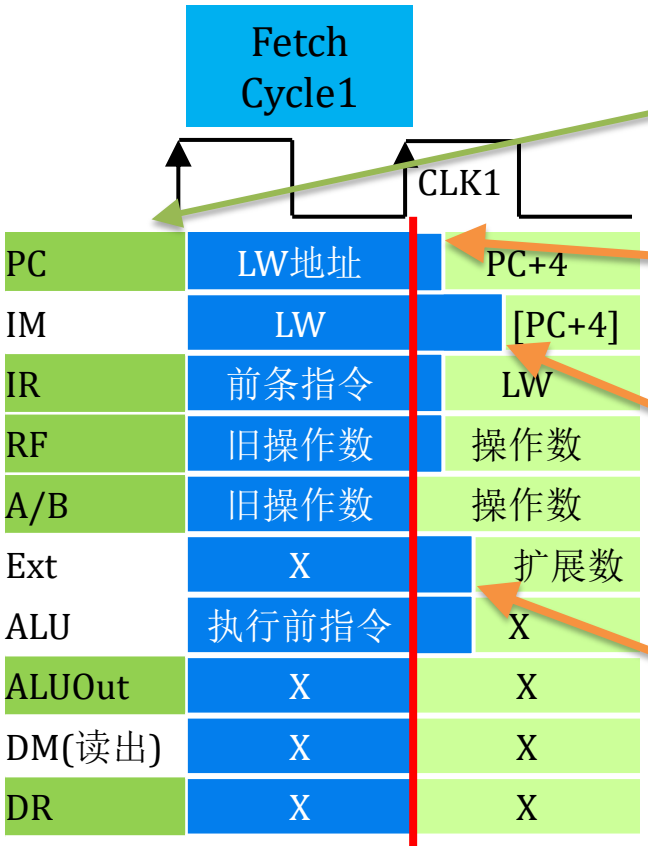
- 建立RTL描述表
  - EXT：这是一个特殊部件，未设置相应的寄存器

周期	步骤	语义	RTL	需控制的功能部件	功能部件控制信号
Cycle1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr←1; NPCOp←+4; PCWr←1
Cycle2	DCD/RF 读操作数	基地址存入A； 偏移符号扩展	A←RF[rs] EXT(Imm16)	EXT	EXTOp←SE
Cycle3	MA 计算地址	执行加法，结果存入ALUOut	ALUOut←ALU(A,EXT)	ALU	EXTOp←SE ALUOp←ADD
Cycle4	MR 读存储器	读取DM，数据存储DR	DR←DM[ALUOut]		
Cycle5	MemWB 回写	DR写入rt寄存器	RF[rt]←DR	RF	RFWr←1

LW时序分析：Cycle1

- Cycle1：取指令(公共周期)
  - 读取IM至IR；同时PC自增4
- X：代表其值无意义
- 边沿后的颜色：代表延迟

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



绿色：功能部件为寄存器

寄存器输出延迟最短

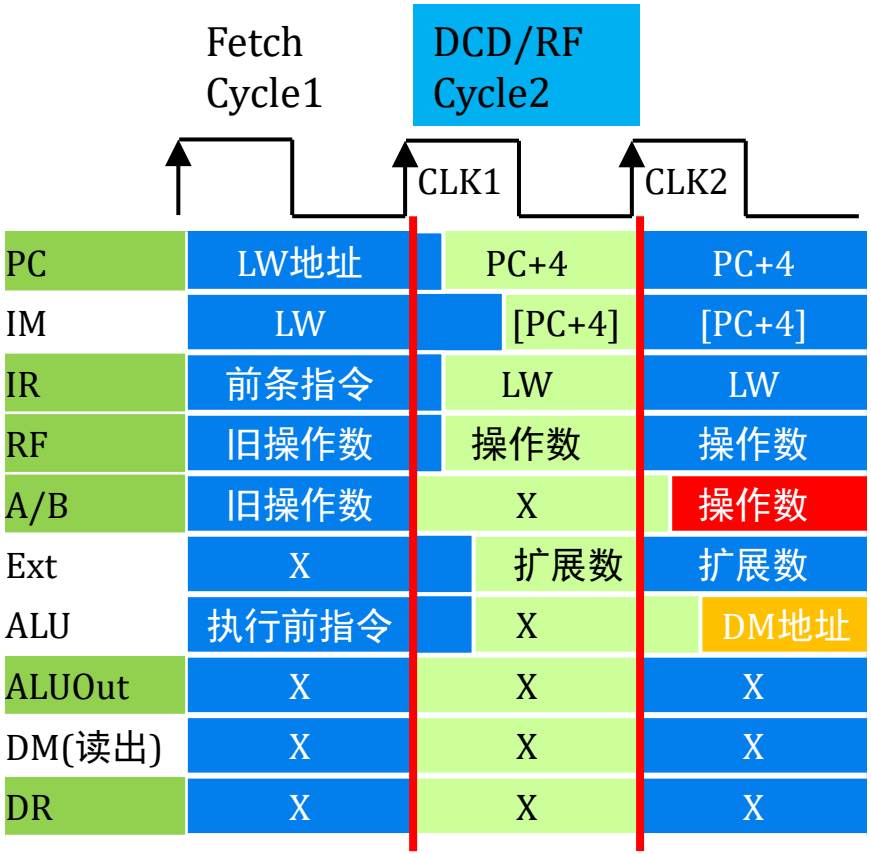
存储器输出延迟比较长

功能部件输出延迟 =
寄存器输出延迟 + 扩展延迟

LW时序分析：Cycle2

- Cycle2：读操作数(公共周期)
  - 读操作数，同时开始译码
    - 指令已经在IR中了

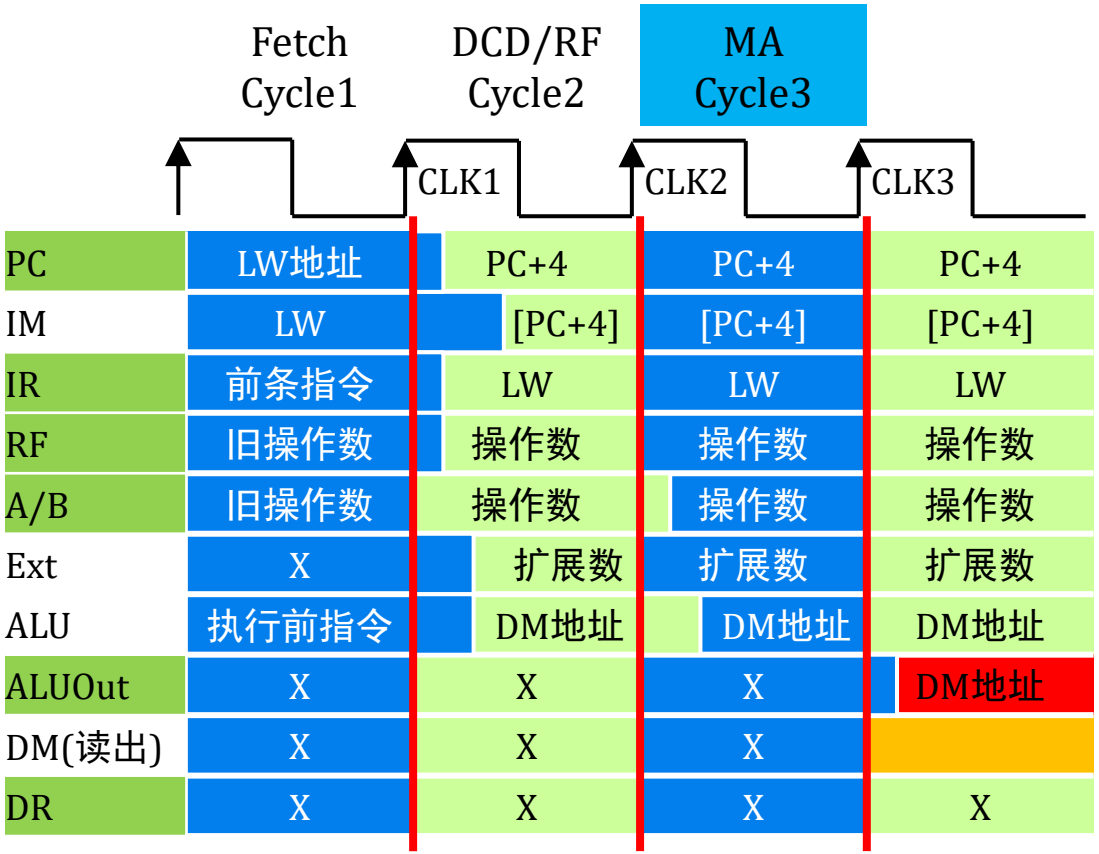
周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



LW时序分析：Cycle3

- Cycle3：计算地址
  - 计算地址并存入ALUOut

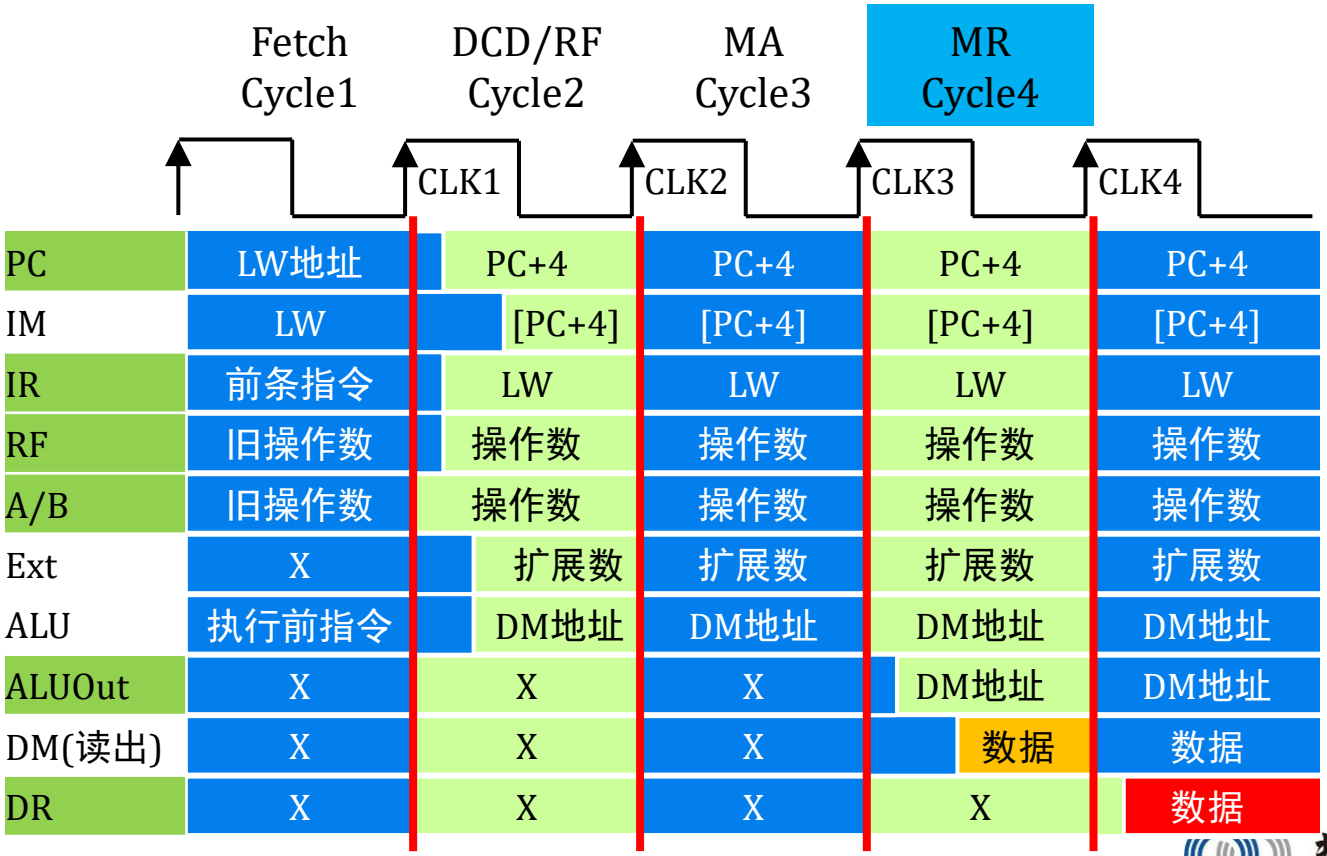
周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



LW时序分析：Cycle4

- Cycle4：读存储器
  - ALUOut驱动DM，数据写入DR

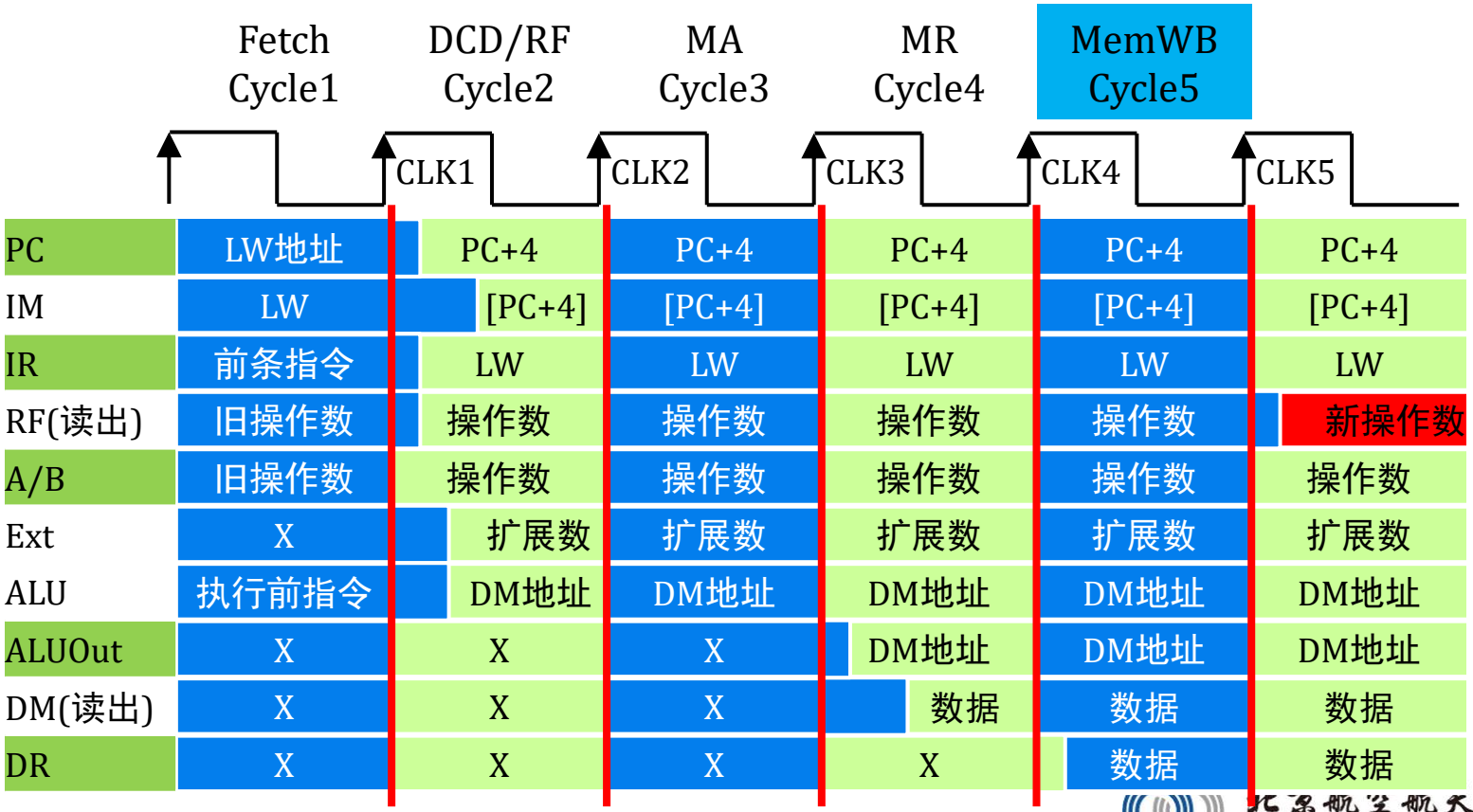
周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



LW时序分析：Cycle5

- Cycle5：写寄存器
  - DM读出数据写入RF

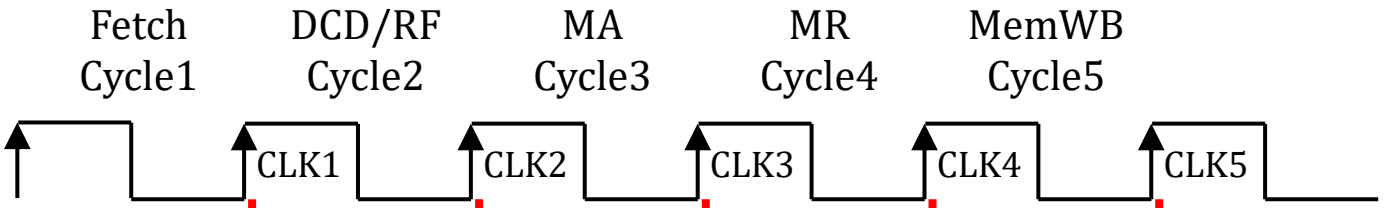
周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



LW时序分析：简明版

- 可以不考虑延迟
- 只关注前后依赖关系

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



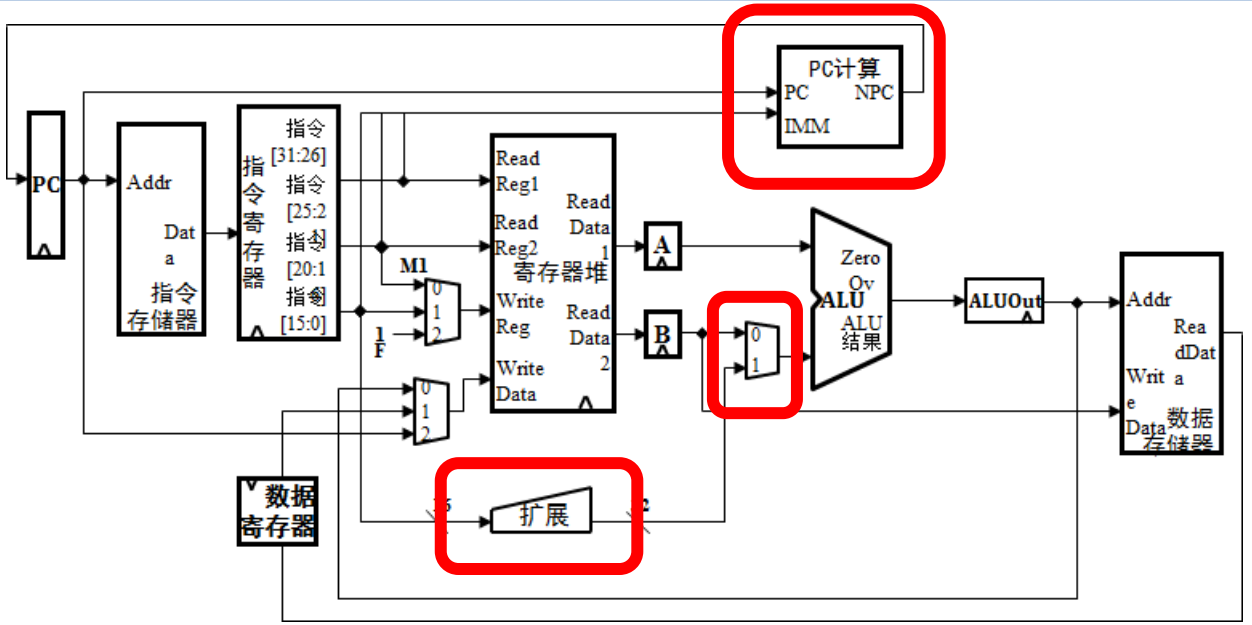
PC	LW地址	PC+4	PC+4	PC+4	PC+4	PC+4
IM	LW	[PC+4]	[PC+4]	[PC+4]	[PC+4]	[PC+4]
IR	前条指令	LW	LW	LW	LW	LW
RF(读出)	旧操作数	操作数	操作数	操作数	操作数	新操作数
A/B	旧操作数	操作数	操作数	操作数	操作数	操作数
Ext	X	扩展数	扩展数	扩展数	扩展数	扩展数
ALU	执行前指令	DM地址	DM地址	DM地址	DM地址	DM地址
ALUOut	X	X	X	DM地址	DM地址	DM地址
DM(读出)	X	X	X	数据	数据	数据
DR	X	X	X	X	数据	数据

目录

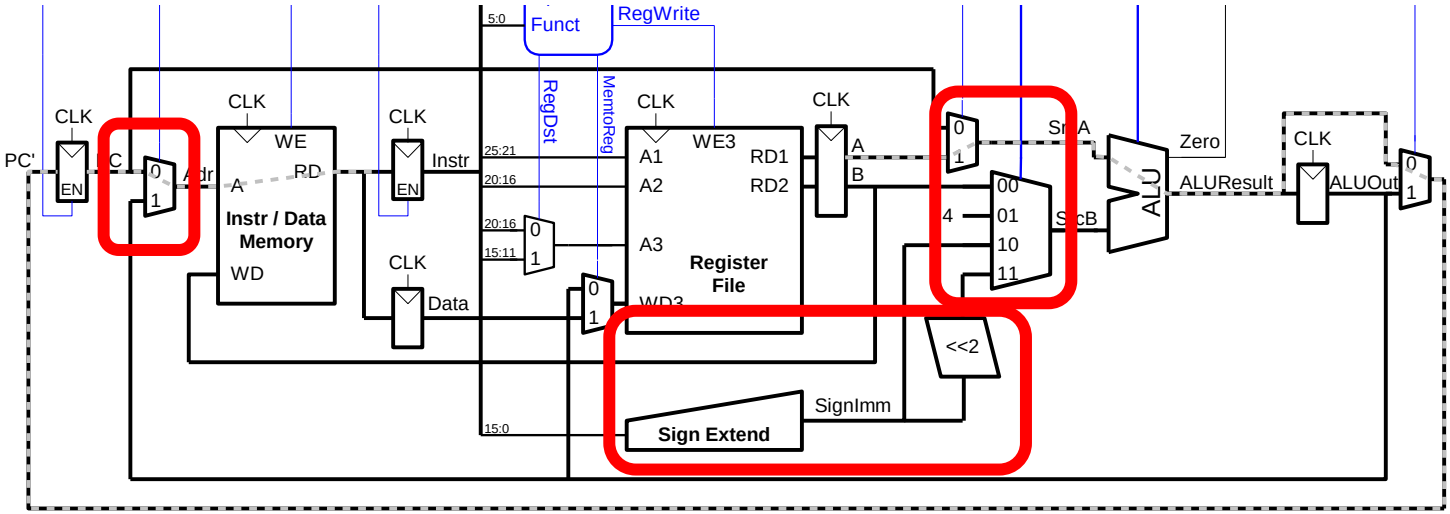
- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

PPT数据通路 vs 教科书数据通路

PPT
数据通路



教科书
数据通路



PPT数据通路 vs 教科书数据通路

- PPT：引入NPC，数据通路的结构化特征更好
  - 整个设计更加简洁，体现高内聚低耦合的设计思想
  - 所有与地址产生的功能都放在NPC中（包括今后的c处理器上电启动地址和异常地址）
  - IM地址端，ALU的A端：没有MUX
- 教科书：可节省一点逻辑，但清晰度不好
  - 只有1个ALU，节省了NPC里的加法器
  - 状态机设计更加复杂，更有利于学习“状态机”